Piazzi . Maldonado . Oliveira

APROFUNDANDO-SE NO

APROFUNDANDO-SE NO MSX

Autores:

Organização Editorial: Pierluigi Piazzi
Coordenação Técnica: Milton Maldonado Jr.
Renato da Silva Oliveira
Co-Autores: Flavio Rossini
Henrique de Figueredo Luz
Paulo Eduardo G. Rossetto
Rubens Pereira Silva Jr.

5ª edição

AGRADECIMENTOS:

Agradecemos aos senhores:

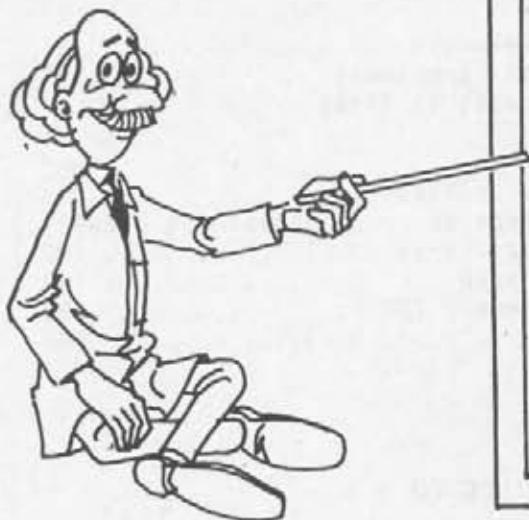
Moris Arditti
João Luiz Terreiro
Júlio Amâncio de Souza
José Mario F. de Andrade
Jairo Zeitel
Marcelo Araújo Muzzillo
Mario Sergio de Oliveira Fontes
Eduardo Toledo Santos
Fernando da Costa Grossi

Pela importante colaboração para a elaboração desta obra.

Direção Editorial : Pierluigi Piazzi
Coordenação Pedagógica: Betty Fromer Piazzi
Produção: Rosa Kogan Fromer
Copy-desk e revisão: Regina Britto Assumpção
Diagramação em computador: Glauter Fabiano Mikahil
Arte final e ilustrações técnicas: Ana Lúcia Antico
Ilustrações: Odilon D. Nicoletti

ALEPH
Publicações e Assessoria Pedagógica Ltda.
Av. Dr. Luis Migliano 1110
05750 São Paulo - SP (011) 843-3202
Caixa Postal 20.707 - CEP 01498

Dados de Catalogação na Publicação (CIP) Internacional
(Câmara Brasileira do Livro, SP, Brasil)

Maldonado Junior, Milton, 1966-
M211a Aprofundando-se no MSX / Mílton Maldonado Jr. et al. -- São Paulo : Aleph, 1986.

1. BASIC (Linguagem de programação para computadores) 2.Linguagens de programação (Computadores) 3. MSX (Computador) - Programação I. Título.

86-0406 CDD-001.642
-001.6424

Índices para catálogo sistemático:
1. BASIC: Linguagem de programação : Computadores : Processamento de dados 001.6424
2. Linguagem de programação : Computadores : Processamento de dados 001.6424
3. MSX : Computadores : Programação : Processamento de dados 001.6424
4. Programação : Linguagem : Computadores : Processamento de dados 001.6424

```
10 COLOR15,4:KEYOFF:WIDTH39:CLS:LOCATE7
,0:PRINT"APROFUNDANDO-SE NO MSX":LOCATE
2,2:PRINT"SUMARIO":LOCATE0,4:PRINT"CAPI
TULO 0  ARQUITETURA DO MSX......009"
20 LOCATE0,6:PRINT"CAPITULO 1  RAM-Mem.
para o usuario..019":LOCATE0,8:PRINT"CA
PITULO 2  ROM-Sist.Oper.do MSX....051"
30 LOCATE0,10:PRINT"CAPITULO 3  PPI-Int
erf.de Perif.....077":LOCATE0,12:PRINT"
CAPITULO 4  VDP-Process.de Video....089
":LOCATE0,14:PRINT"CAPITULO 5  PSG-Proc
ess.de Sons.....109"
40 LOCATE0,16:PRINT"CAPITULO 6  Disposi
tivos de Armaz...121":LOCATE0,18:PRINT"
APENDICE I   Redef.de Caracteres....147
":LOCATE0,19:PRINT"APENDICE II  Sistema
s de Numeracao..149"
50 LOCATE0,20:PRINT"APENDICE III Intr.a
o Assembly Z80...153":LOCATE0,21:PRINT"
APENDICE IV  Rotinas do BIOS........156
"
60 GOTO60
```

SUMÁRIO

PREFÁCIO PARA A QUARTA EDIÇÃO

Quando escrevemos o prefácio para a primeira edição deste livro, lançada no início de 1986, o padrão MSX ainda era uma promessa agradável que já estava gerando uma enorme curiosidade no mundo da micro-informática no Brasil. A prova disto foi a rapidez com a qual se esgotaram as três primeiras edições deste livro.

Enquanto preparávamos a próxima edição, muitas coisas foram acontecendo: recebemos inúmeras sugestões, críticas, elogios e alertas por parte dos leitores que permitiram aperfeiçoar ainda mais esta obra, eliminando erros e acrescentando informações. Um tipo particular de elogio nos tocou de maneira especial : num país onde "esconder o leite" (em todos os sentidos!) é considerado uma grande esperteza, o fato de termos "contado tudo" foi considerado altamente elogiável e porisso queremos agradecer aos leitores que perceberam a importância desta nossa atitude.

Outro fato importante foi a busca, por parte dos grandes fabricantes, de um grau de padronização cada vez maior. A instituição de um padrão BRASCII para os caracteres das impressoras, realizada pela ABNT (Associação Brasileira de Normas Técnicas) foi um passo gigantesco no sentido de se eliminar a terrível "Babel" de tabelas que domina o mundo da informática. A eliminação de um problema no futuro, porém, gerou um problema presente pois temos um grande parque de impressoras e micros que foram fabricados antes desta decisão. Para orientar seus usuários, acrescentamos ao livro PROGRAMAÇÃO AVANÇADA EM MSX uma série de programas em linguagem de máquina que permitem fazer esta compatilização.

Finalmente devemos citar outro acontecimento, talvez o mais importante de todos: a fabricação de disk drives para o padrão MSX. Isto abre as portas de um universo de utilizações profissionais para os micros do padrão MSX, permitindo o uso imediato de uma enorme quantidade de software já desenvolvido, especialmente em CP/M.

Para os usuários deste importante periférico publicamos a obra USANDO O DISK DRIVE NO MSX, escrito por Rubens Pereira Silva Jr, um dos co-autores deste livro.

Até o final de 1986, já tínhamos publicado 13 títulos referentes ao MSX. Isto, unido ao esforço de outras editoras e "softhouses" faz com que os 100.000 usuários de MSX tenham amplas condições de utilização desta máquina maravilhosa que, no Brasil, veio definitivamente para ficar.

Bem vindos ao clube!

NOTA DO EDITOR

Turing foi um matemático inglês que realizou estudos em Cibernética. É dele o famoso critério, conhecido por todos os que lidam com Inteligência Artificial:

"Imagine algo trancado numa sala que você não possa ver mas com o qual você possa dialogar através de um terminal (tipo telex). Se, após um tempo suficientemente longo,você não conseguir descobrir se trata-se de uma máquina ou de um ser humano, com certeza o que está trancado na sala é inteligente".

Quem assistiu 2001, por exemplo, percebe que HAL, o computador de bordo, passaria tranqüilamente nesse teste.

Nossos micros pessoais, porém, estão muito longe de serem aceitos como inteligentes pelo critério de Turing. Existe um componente do micro, entretanto, que passaria tranquilamente pelo teste: o usuário!

Este livro foi escrito justamente para ele, a única parte realmente inteligente do microcomputador.

Quando este livro foi elaborado, para atender a uma grande quantidade de usuários sedentos de informações, pensou-se num leitor-alvo com as seguintes características:

1) Bom conhecedor de Basic, se não do MSX pelo menos de algum dialeto anterior ao MSX, tipo SINCLAIR, TRS-80, APPLE ou IBM-PC.

2) Inteligente (pelo menos segundo o critério de Turing) e curioso.

3) Ignorante em quase tudo que se refere ao padrão MSX (senão este livro não seria necessário!)

4) Ignorante (ou quase) em linguagem de máquina. Logicamente algum co-

nhecimento de Assembly vai ajudar, mas não é indispensável.

Ao terminar este livro, o leitor se tornará agente de um processo teorizado por outro genial matemático: A MÁQUINA DE VON NEUMANN.

Para entendermos o que é uma máquina de Von Neumann vamos pensar num exemplo teórico.

Imagine o leitor que haja necessidade de se escavar uma enorme montanha de minério! A tarefa vai ser dividida entre dois engenheiros dispondo de verbas ilimitadas, que vão atacar a tarefa de lados opostos.

De um lado temos o engenheiro tecnocrata que vai, em primeiro lugar, fazer um estágio nos Estados Unidos para ver como se desmontam montanhas e que equipamentos importar.

Obviamente será fundada a Montanhabrás, serão definidos organogramas, fluxogramas e outros gramas. Será montada uma fábrica de tratores, com uma produção de centenas por mês e os trabalhos de terraplenagem serão iniciados a todo vapor, para cavar o lado tecnocrata da montanha.

Do outro lado temos o engenheiro criativo que, além de estudar terraplenagem se diverte com cibernética.

Ele projeta e fabrica um único trator, desengonçado, lento mas que usa a matéria escavada da montanha para construir outro trator, idêntico ao original. Esta é uma máquina de Von Neumann.

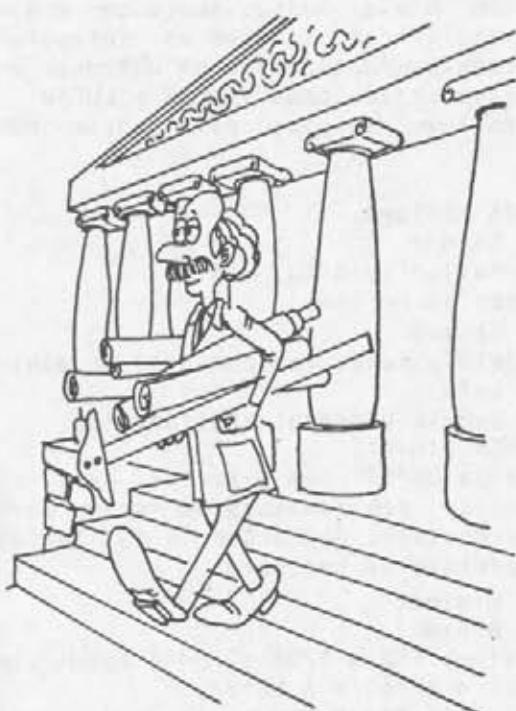
É claro que, no lado de cá da montanha, os trabalhos começarão num ritmo muito mais lento que do lado tecnocrata.

Para um observador desprevinido, olhando apenas o processo inicial, o lado cibernético perderá a corrida!

Existirá um dia, porém, em que a progressão geométrica das máquinas de Von Neumann, equilibrará a progressão aritmética dos tratores tecnocratas. A partir deste momento, o atraso será eliminado a passos gigantescos e a quase totalidade da montanha será engolida pelas máquinas de Von Neumann!

Como o leitor já deve ter percebido, os seres vivos, incluindo o próprio Homem, são um caso particular de máquinas de Von Neumann (o monolito do 2001 é outro tipo, mas isso já é outra conversa).

Agora surge a pergunta: em que sentido o leitor deste livro se transformará numa dessas máquinas?

A resposta é simples: se a nossa equipe usasse todo o "know-how" que tem para produzir software para o MSX, o usuário de nossos produtos seria um simples consumidor.

Usando o mesmo "know-how" para escrever um livro, onde são explicados os truques de funcionamento e programação, o usuário torna-se um produtor, alguém capaz de levar adiante o conhecimento que lhe foi transmitido.

Esta é uma verdadeira forma de se informatizar uma sociedade, eliminando o nosso atraso tecnológico: transformar o consumidor em produtor. A progressão geométrica assim formada é irreversível, é uma verdadeira bola de neve de crescimento exponencial.

Há mais programadores auto-didatas no Brasil (que aprenderam com livros em cima de modestos computadores domésticos) do que os pomposamante formados por todas as escolas de processamento de dados no país!

Nossa equipe poderia receber a remuneração de seu trabalho tanto produzindo software, quanto produzindo livros.

Preferimos o segundo caminho: criar máquinas de Von Neumann é muito mais gratificante e, no fundo, muito mais útil para o Brasil.

Capítulo 0

ARQUITETURA DO MSX

Apresentação

O padrão MSX nasceu de uma associação entre a Microsoft americana com algumas das maiores empresas japonesas no ramo da eletrônica digital. Usando um dos melhores microprocessadores de oito bits fabricados atualmente e a experiência adquirida nos últimos dez anos em hardware e software, surgiu um microcomputador com qualidades suficientes para tornar-se um sucesso mundial: o micro padrão MSX.

A linha MSX une uma arquitetura de hardware arrojada com um BASIC sofisticado e bastante rápido, além de oferecer uma vantagem a mais para o usuário: a padronização. Não importando a marca, qualquer MSX possui a mesma arquitetura.

O usuário que adquire um MSX tem a certeza que poderá usar um software ou periférico desenvolvido para a linha MSX em qualquer parte do mundo e para qualquer outra marca.

fig. 0.1 - Algumas empresas do "clube MSX".

Empresa	País	Empresa	País	Empresa	País
SONY	Japão	CASIO	Japão	MITSUMI	Japão
SANYO	Japão	FUJITSU	Japão	KYOCERA	Japão
NATIONAL	Japão	YAMAHA	Japão	PHILIPS	(Europa)
PIONEER	Japão	JVC	Japão	GOLDSTAR	Cor. Sul
MITSUBISHI	Japão	CANON	Japão	SPECTRAVIDEO	E.U.A.
HITACHI	Japão	TOSHIBA	Japão	DAEWOO	Coréia do Sul
GRADIENTE	Brasil	até 02/1987			
EPCOM	Brasil				

O apoio e o suporte tecnológico à linha MSX é fornecido por gigantes da eletrônica mundial (veja na figura 0.1 a relação de algumas empresas fabricantes de micros MSX). Com o aval dessas empresas, o padrão MSX estimula toda a indústria de periféricos e também as software-houses, pois a certeza de que seus produtos funcionarão em qualquer equipamento MSX do mundo é rapidamente visualizada como VENDAS + LUCRO.

Entre as principais características exigidas pelo padrão MSX destacamos as seguintes:

* Microprocessador Central	- Z80A (Zilog)
* Freqüência de clock	- -3,58 MHz
* ROM	- 32 Kbytes (padrão, mínimo)
* RAM	- 8 Kbytes (mínimo)
* RAM de Vídeo (VRAM)	- 16 Kbytes
* Microprocessador de Vídeo (VDP)	- TMS9128A(Texas Instruments) ou equivalente
* Microprocessador de Som (PSG)	- AY-3-8910 (General Instruments)
* Circ. Aux. de Comunicação (PPI)	- 8255A (Intel)
* Teclado	- padrão QWERTY com 5 teclas para 10 funções programáveis, 4 teclas para movimentação do cursor e 13 teclas especiais de controle.
* SLOT's para cartuchos	- 1 (mínimo)
* Entrada para joystick	- 1 (mínimo)
* Acesso a gravador cassete	- Sistema FSK a 1200 e 2400 bauds com controle sobre o motor.
* Acesso a disk-drive	- Via DISK-BASIC MSX (contido na própria ROM) ou via MSX-DOS.
* Acesso à impressora	- Padrão paralelo Centronics.
* Caracteres	- Caracteres padrão MSX (inclui os códigos ASCII) obrigatórios.

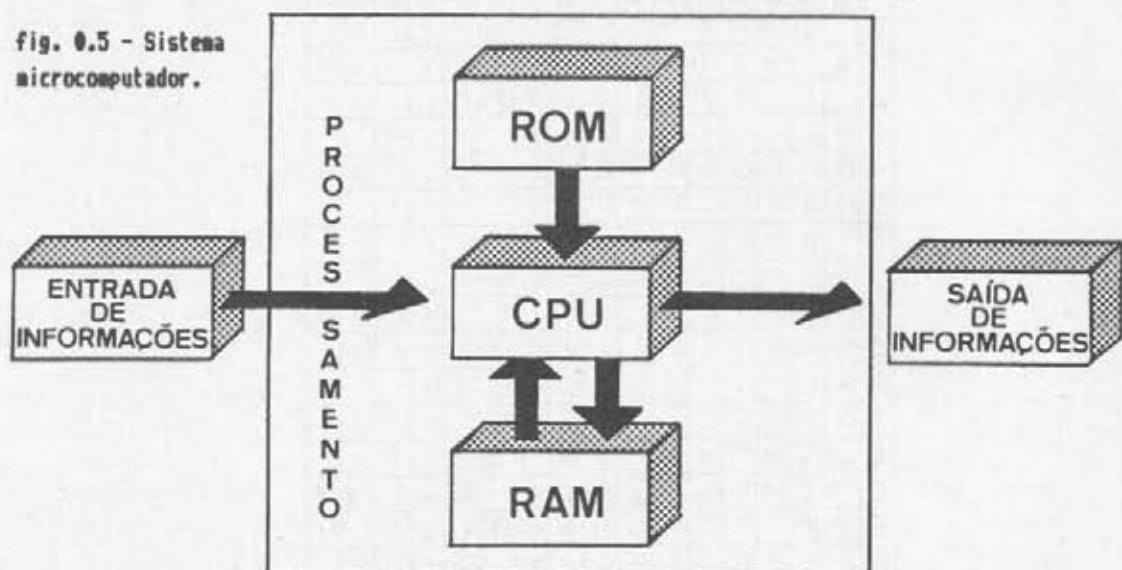
De fabricante para fabricante, algumas características podem variar: o número de slots para cartucho, o número de entradas para joystick, a quantidade de bytes da ROM e da RAM, parte da tabela de caracteres, etc. O padrão básico de hardware e software, entretanto, continua o mesmo.

fig. 0.2 - Tabela de caracteres da 1ª versão do MSX Expert.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		☺	☻	♥	♦	♣	♠	·	◘	○	◙	♂	♀	♪	♫	☼
1	┼	┴	┬	┤	├	┼	│	─	┌	┐	└	┘	╳	╱	╲	+
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	ç	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	Ç	▲
8	~	ü	é	â	ä	à	å	`	ê	ë	è	ï	î	ì	Ä	Å
9	É	æ	Æ	ô	ö	ò	û	ù	ÿ	Ö	Ü	¢	£	¥	₧	ƒ
A	á	í	ó	ú	ñ	Ñ	ª	º	¿	⌐	¬	½	¼	¡	«	»
B	Ã	ã	Ĩ	ĩ	Õ	õ	Ũ	ũ	Ĳ	ĳ	¾	∽	◊	‰	¶	§
C	▁	▚	▆	▔	▪	▀	▏	▞	▊	▕	▉	▨	▧	▼	▲	▶
D	◀	⧗	⧓	▘	▗	▝	▖	※	Δ	‡	ω	█	▄	▌	▐	▀
E	α	β	Γ	π	Σ	σ	µ	γ	Φ	θ	Ω	δ	∞	ø	∈	∩
F	≡	±	≥	≤	⌠	⌡	÷	≈	°	∙	·	√	^{n}	^{2}	▮	

fig. 0.3 - Tabela de caracteres da 1ª versão do MSX HOTBIT.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		☺	☻	♥	♦	♣	♠	•	◘	○	◙	♂	♀	♪	♫	☼
1	┼	┴	┬	┤	├	┼	│	─	┌	┐	└	┘	╳	╱	╲	╋
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	~	⌂
8	Ç	ü	é	â	Á	à	¨	ç	ê	í	ó	ú	Â	Ê	Ô	À
9	É	æ	Æ	ô	ö	ò	û	ù	ÿ	Ö	Ü	¢	£	¥	₧	ƒ
A	á	í	ó	ú	ñ	Ñ	ª	º	¿	⌐	¬	½	¼	¡	«	»
B	Ã	ã	Ĩ	ĩ	Õ	õ	Ũ	ũ	Ĳ	ĳ	¾	∼	◇	‰	¶	§
C	▁	▚	▄	▔	▪	▀	▎	▞	▌	▕	█	▨	▧	▼	▲	▶
D	◀	⧗	⋈	▘	▗	▝	▖	▩	Δ	ǂ	ω	█	▄	▌	▐	▀
E	α	β	Γ	π	Σ	σ	µ	γ	Φ	θ	Ω	δ	∞	φ	∈	∩
F	≡	±	≥	≤	⌠	⌡	÷	≈	°	∙	·	√	^{n}	^{2}	■	

fig. 0.4 - Tabela de caracteres da 2ª versão dos MSX HOTBIT e Expert.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		☺	☻	♥	♦	♣	♠	•	◘	○	◙	♂	♀	♪	♫	☼
1	┼	┴	┬	┤	├	┼	│	─	┌	┐	└	┘	╳	╱	╲	╋
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	~	⌂
8	Ç	ü	é	â	Á	à	¨	ç	ê	í	ó	ú	Â	Ê	Ô	À
9	É	æ	Æ	ô	ö	ò	û	ù	ÿ	Ö	Ü	¢	£	¥	₢	ƒ
A	á	í	ó	ú	ñ	Ñ	ª	º	¿	⌐	¬	½	¼	¡	«	»
B	Ã	ã	Ĩ	ĩ	Õ	õ	Ũ	ũ	Ĳ	ĳ	¾	∼	◇	‰	¶	§
C	▁	▚	▄	▔	▪	▀	▎	▞	▌	▕	█	▨	▧	▼	▲	▶
D	◀	⧗	⋈	▘	▗	▝	▖	▩	Δ	ǂ	ω	█	▄	▌	▐	▀
E	α	β	Γ	π	Σ	σ	µ	γ	Φ	θ	Ω	δ	∞	φ	∈	∩
F	≡	±	≥	≤	⌠	⌡	÷	≈	°	∙	·	√	^{n}	^{2}	■	

OBS: A tabela de caracteres da versão 2, tanto do Expert quanto do HOTBIT, está de acordo com a padronização estabelecida pela ABNT (Associação Brasileira de Normas Técnicas). A única diferença ainda existente está no caractere &HF0, que no HOTBIT permanece como na versão 1.

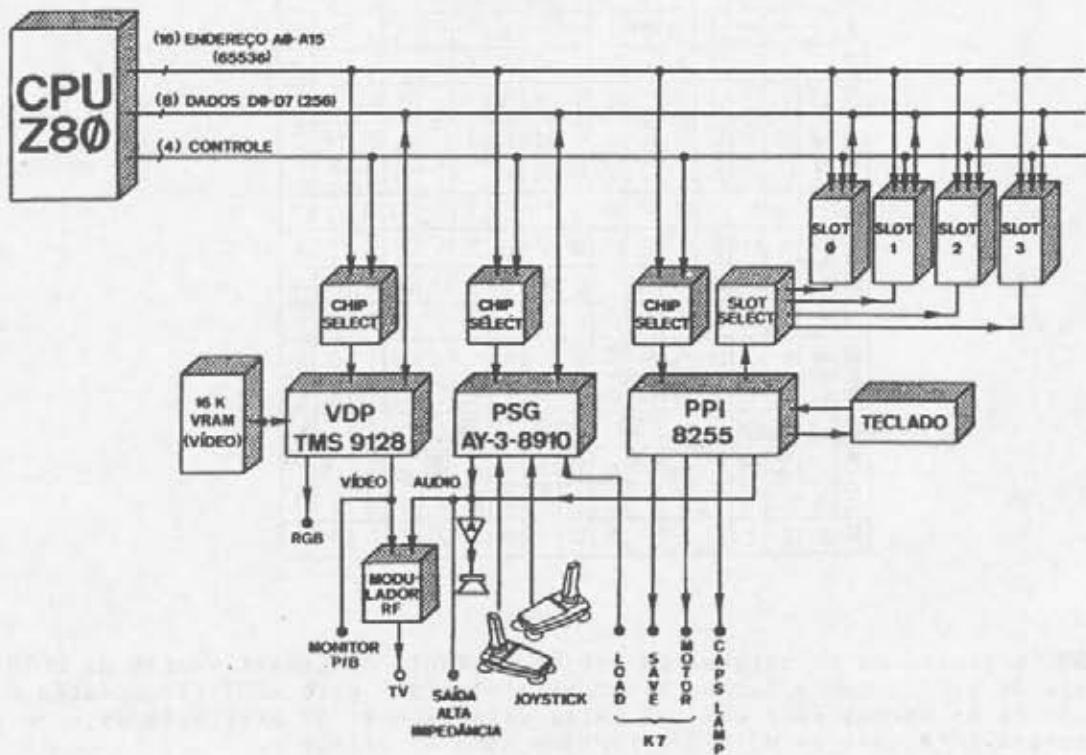
A estrutura lógica de um microcomputador

Podemos representar um sistema de microcomputador esquematicamente e de forma simplificada de acordo coma a figura 0.5.

fig. 0.5 - Sistema microcomputador.

No caso dos micros MSX, a ENTRADA de dados é feita via teclado e/ou gravador cassete, disk-driver, modem, joysticks, paddles ou light pen.

fig. 0.6 - Diagrama simplificado da estrutura lógica do MSX.

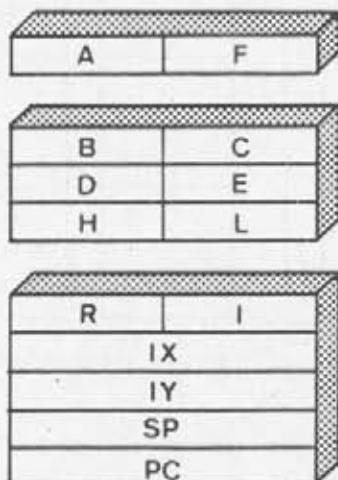
CPU Z80
(16) ENDEREÇO A0-A15 (65536)
(8) DADOS D0-D7 (256)
(4) CONTROLE
SLOT 0
SLOT 1
SLOT 2
SLOT 3
CHIP SELECT
CHIP SELECT
CHIP SELECT
SLOT SELECT
16 K VRAM (VÍDEO)
VDP TMS 9128
PSG AY-3-8910
PPI 8255
TECLADO
VÍDEO
AUDIO
RGB
MODULADOR RF
MONITOR P/B
TV
SAÍDA ALTA IMPEDÂNCIA
JOYSTICK
LOAD
SAVE
MOTOR
CAPS LAMP
K7

O PROCESSAMENTO é realizado pelo Z80A, usando o Sistema Operacional da ROM e o programa de RAM.

A SAÍDA de informações se dá, geralmente, através de um vídeo (monitor ou TV) ou de uma impressora, mas também pode ser através de um gravador cassete, de um disk-driver, de um plotter ou de um modem. Há, ainda, a saída sonora através de um alto-falante interno ou do terminal externo ÁUDIO, presente em algumas máquinas

Dependendo das peculiaridades de cada fabricante, os meios de Entrada/Saída de dados podem ser outros, além dos citados.

O Z80A (CPU) é auxiliado por três outros circuitos que se encarregam de grande parte das funções de entrada e saída de informações. O 8255A (PPI), o TMS9128A (VDP) com VRAM e o AY38910 (PSG) (fig. 0.6).

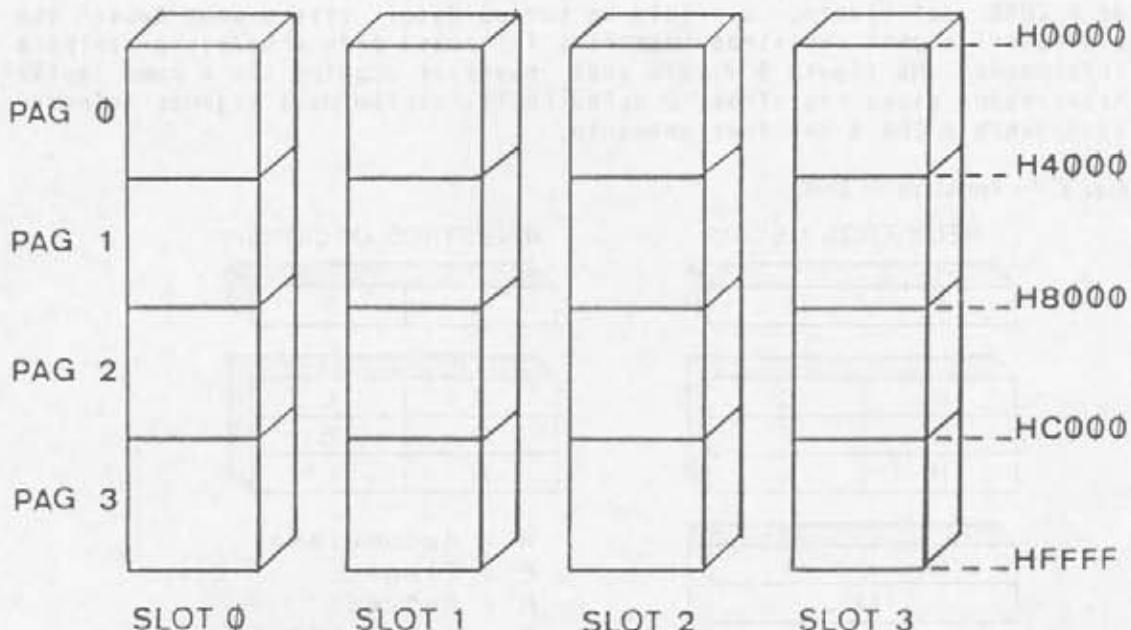
A Unidade Central de Processamento (CPU)

A parte mais importante de um micro computador é sua CPU. No MSX ela é um microprocessador Z80A da fábrica Zilog funcionando com uma freqüência de clock por volta de 3,58 MHz (isto é, em 1 segundo, ela é capaz de executar quase um milhão de operações elementares!).

O Z80A é um microprocessador de 8 bits (ou 1 byte) e, portanto, pode processar 8 bits de cada vez. O endereçamento de memória, entretanto, é feito com 16 bits (ou 2 bytes). Com 16 bits pode-se produzir 65536 configurações diferentes (ou 64 Kbytes) e é esse, portanto, o número de bytes que o Z80A pode controlar diretamente.

Não nos interessa, especificamente agora, conhecer mais a fundo o Z80A, entretanto, a título de curiosidade, vale a pena saber que ele possui alguns registros (memórias internas) onde armazena e manipula informações. Na figura 0.7 você pode observar quantos são e como estão organizados esses registros. O apêndice III contém mais algumas informações sobre o Z80 e seu funcionamento.

fig. 0.7 - Registros do Z80A.

Organização da memória

Uma das características que tornam o padrão MSX extremamente versátil é a estrutura de organização de sua memória. Apesar de poder controlar simultaneamente apenas 64 Kbytes de memória, no MSX a CPU pode trabalhar com até 1 Mbyte (1024 Kbytes).

Através de um sistema de chaveamento, a CPU escolhe o sub-conjunto de 64 Kbytes que irá controlar e, durante o processamento, pode ir mudando de sub-conjunto.

As vantagens desse tipo de arquitetura são evidentes e serão melhor compreendidas nos próximos itens.

Slots e páginas

Podemos imaginar a memória do MSX dividida em 4 partes principais, ou slots, que chamaremos de SLOT 0, SLOT 1, SLOT 2 e SLOT 3. Cada um desses slots pode conter 64 Kbytes, divididos também em 4 partes, ou páginas, chamadas PÁGINA 0, PÁGINA 1, PÁGINA 2 e PÁGINA 3. (fig. 0.8).

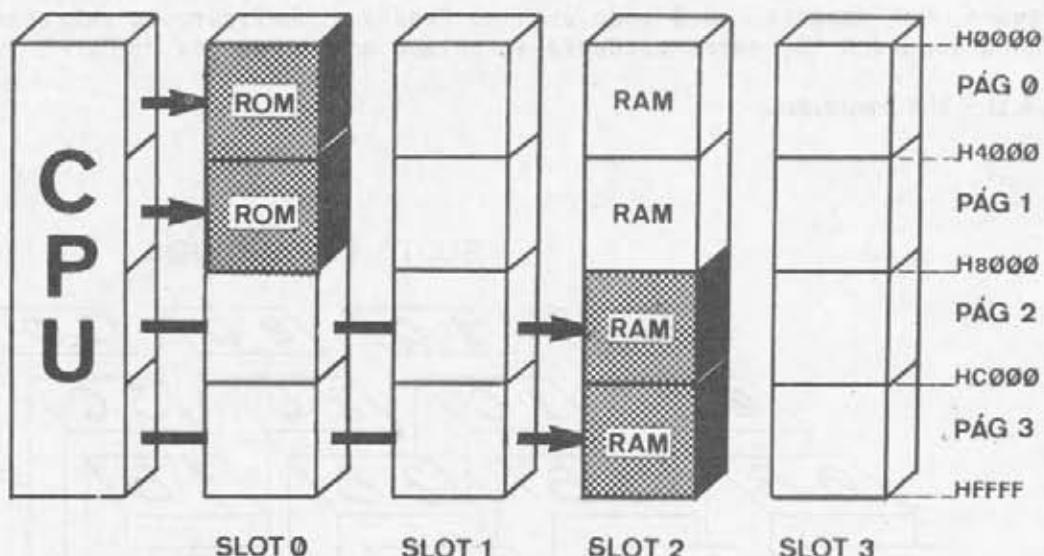



fig. 0.8 - Slots e páginas.

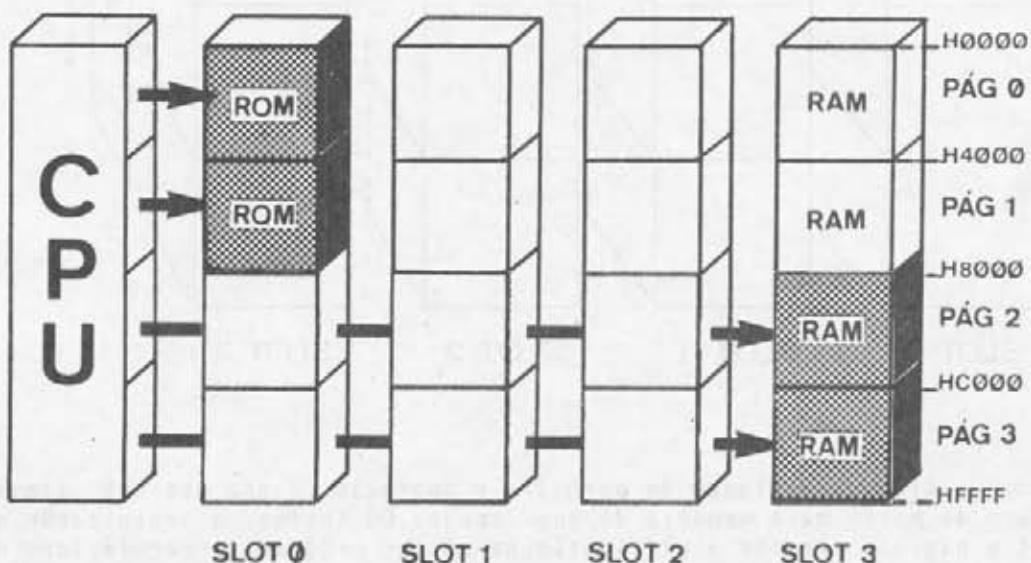
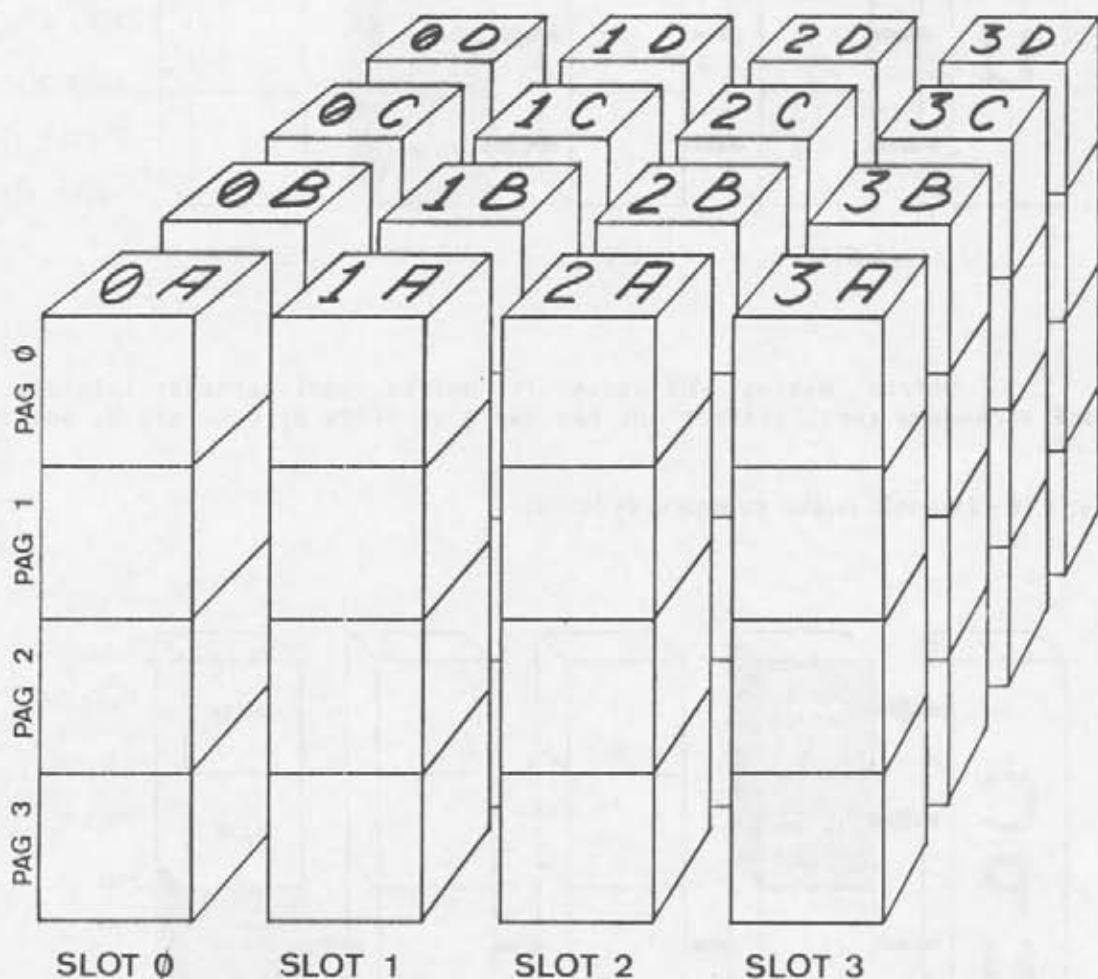
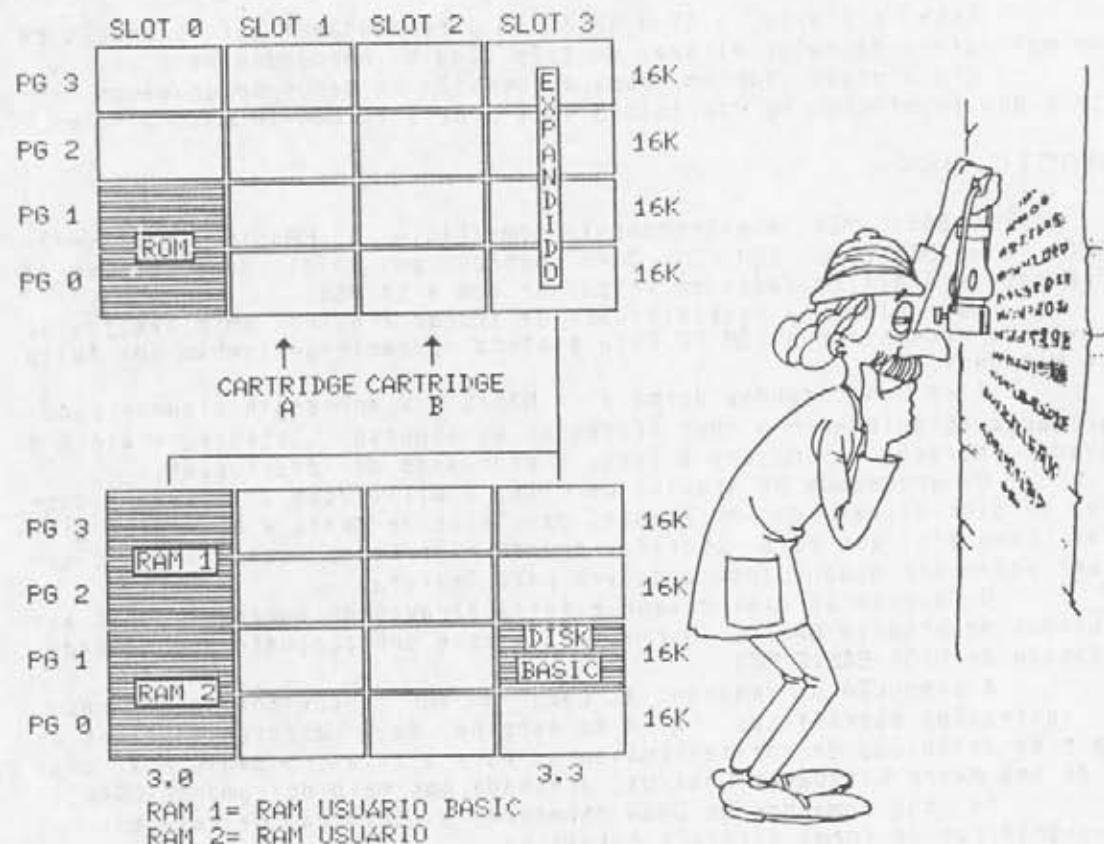
Cada página de memória tem 16 Kbytes e seus endereços são sempre os mesmos. Por exemplo, a página 2 de qualquer slot estará sempre entre os endereços &H8000 e &HBFFF. É por isso que nunca se pode usar duas páginas de mesmo número simultaneamente.

Através de software pode-se escolher um conjunto qualquer de 4 páginas (de 0 a 3). Por exemplo, o EXPERT da Gradiente vem com uma configuração inicial que usa as páginas 0 e 1 do slot 0 (para a ROM) e as páginas 2 e 3 do slot 2 (para a RAM), como mostra a figura 0.9.

fig. 0.9 - Paginação inicial da memória do Expert.

Outros micros MSX podem ter outras configurações iniciais. Você aprenderá como "chavear" as páginas e os slots no capítulo 3. sobre a PPI.

fig. 0.10 - Paginação inicial da memória do HOTBIT

Com 4 slots de 64 Kbytes cada, temos um total de 256 Kbytes. Para operar com 1 Mbyte (4x256 Kbytes), cada slot do MSX pode ser expandido. Veja a figura 0.11 .

A escolha das 4 páginas continua a ser factível através de software. Por exemplo, você pode usar no Expert a configuração representada na figura 0.10, correspondente ao estado normal do MSX HOTBIT.

fig. 0.11 - Slot expandidos.

Além da vantagem de permitir a operação, ainda que não simultânea, de muito mais memória do que apenas 64 Kbytes, a organização em slots e páginas permite a utilização de vários programas, periféricos ou mesmo várias linguagens.

Você pode ter, por exemplo, um compilador FORTRAN carregado nas páginas 0 e 1 do slot 1A, um compilador COBOL nas páginas 2 e 3 do slot 3A e ainda um sistema operacional nas páginas 0 e 1 do slot 2D.

Nos modelos mais recentes do Expert (DD PLUS e PLUS), lançados no final de 1989, a configuração dos slots obedece a uma configuração diferente da adotada inicialmente. O slot 3 já vem expandido em sub-slot 3.0 (3A) e 3.3 (3D). No 3.0 fica a RAM e, na página 1 do sub-slot 3.3 temos a interface de disco (DD PLUS) ou um software residente (PLUS).

Veja no livro "+50 DICAS PARA MSX" como descobrir a configuração dos slots do seu MSX (dica 36).

A PPI (Peripheral Programmable Interface)

A PPI 8255A é um circuito auxiliar que alivia a CPU de algumas tarefas e que será estudada mais detalhadamente no capítulo 3.

As principais tarefas da PPI são:

* Acesso ao teclado;
* Controle do motor do gravador cassete;
* Envio de dados ao gravador cassete;
* Controle da lâmpada do CAPS LOCK (se ela existir!);
* "CLICK" do teclado;
* Selecionamento de páginas e slots de memória.

O VDP (Video Display Processor)

O VDP TMS9128 é o responsável pela geração e pelo controle de imagens. Para armazenar caracteres, imagens, cores e sprites, ele tem

sob seu controle 16 kbytes de RAM, independentes da memória principal (dos slots). A CPU não tem acesso direto a essa RAM de 16 kbytes a não ser através do VDP e, porisso, ela é denominada VRAM (Video-RAM). No capítulo 4, a estrutura e o funcionamento do VDP estão explicados detalhadamente.

O PSG (Programmable Sound Generator)

Esse circuito, o AY-3-8910, é o responsável pela produção de sons musicais e de ruído através de três canais independentes.

Ele é usado, também, para a recepção de dados do gravador cassete e dos joysticks. No capítulo 5 você poderá estudá-lo mais a fundo.

O BASIC MSX

O BASIC MSX é extremamente semelhante ao MBASIC (Microsoft-BASIC) e ao BASIC do IBM PC . Quem conhece uma dessas duas versões do BASIC não terá dificuldades em trabalhar com a do MSX.

Há, ainda, a possibilidade de trocar arquivos em diskettes ou através de modems com o IBM PC (seu sistema operacional também foi feito pela Microsoft!).

Além dos comandos normais, o BASIC MSX apresenta algumas peculiaridades relacionadas a seus processos de arquivo, controle de disco e gravador, geração de imagens e sons, e processos de interrupção.

Os processos de arquivo permitem a utilização do gravador cassete, do disk-driver, da impressora, das telas de texto e das telas gráficas como arquivos para escrita. Apenas o gravador cassete e o disk-driver podem ser usados como arquivos para leitura.

O acesso ao disk-driver é feito através de comandos especiais embutidos no próprio BASIC. Normalmente, esse sub-conjunto de comandos é chamado de DISK-BASIC MSX.

A produção de imagens, a cargo do VDP, é controlada, também, por instruções específicas. Além da escolha de 4 diferentes tipos de tela e da definição de sprites (máscaras para a tela), o BASIC MSX dispõe de uma Macro-Linguagem Gráfica, acessada por meio do comando DRAW.

Os sub-comandos do DRAW permitem a confecção de desenhos nas telas gráficas de forma direta e intuitiva.

A produção de sons, a cargo do PSG, também requer comandos específicos: o PLAY e o SOUND.

O comando PLAY permite a produção de sons musicais através de três canais individuais, usando a notação musical cifrada como sub-linguagem.

O SOUND acessa diretamente os registros do PSG e, através disso, além de sons musicais, pode também gerar ruídos.

O BASIC MSX é interrompido 60 vezes por segundo (de acordo com a rede elétrica). Existem instruções que permitem interromper a execução de um programa e desviá-lo para uma sub-rotina através da digitação de uma certa tecla, da sobreposição de sprites na tela, ou sempre que transcorrer um determinado intervalo de tempo.

Como recursos adicionais, o BASIC MSX dispõe ainda de comandos para acesso à VRAM via VDP (VPEEK e VPOKE), comandos para traçados gráficos (LINE,CIRCLE, etc...), comandos para troca de atributos entre variáveis de mesmo tipo (SWAP e MID$) e instruções especiais para tratamento de dados numéricos (&H, &O, &B, HEX$, OCT$, BIN$, \ e MOD). Além disso, os operadores lógicos (AND, OR, NOT, XOR, EQV e IMP) permitem comparações bit a bit entre dois números em decimal. Por exemplo:

```
PRINT 74 AND 89
```

resultará em 72 !

Capítulo 1

RAM - MEMÓRIA PARA USUÁRIO

Distribuição da RAM

Você sabe que os MSX brasileiros têm 80 Kbytes de memória RAM (Random Access Memory), uma memória que pode ser gravada e/ou lida. É nessa memória que os programas do usuário são introduzidos.

Isso quer dizer que você tem 80 mil posições de memória para armazenar até 80 mil caracteres.

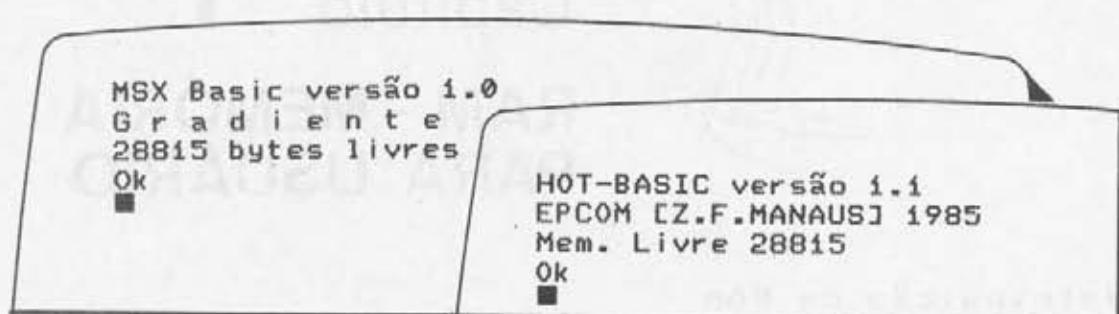
Desses 80 Kbytes de RAM, como já vimos, 16 Kbytes são reservados exclusivamente para o circuito de vídeo. Essa região da memória é chamada VRAM (Vídeo-RAM) e será estudada em detalhes no Capítulo 4.

Restam, portanto, 64 Kbytes de RAM dos 80 que tínhamos disponíveis.

Você já leu, no capítulo 0, que o Z80 só consegue gerenciar 64 Kbytes de memória e que o MSX tem 32 Kbytes de ROM, onde está armazenado o Sistema Operacional e o BASIC. Sendo assim, não podemos ter, ao mesmo tempo, 32 Kbytes de ROM e 64 Kbytes de RAM (o que daria 96 Kbytes de memória). O Z80 não consegue manipular tantos bytes de uma só vez.

Dos 64 Kbytes de memória RAM, apenas 32 estão realmente disponíveis para usarmos com programas em BASIC, ficando os outros 32 restantes "escondidos", de maneira que apenas os cartuchos, os acionadores de diskette (disk-drivers) e programas em Assembly conseguem acessá-los.

À primeira vista pode parecer bobagem instalar 64 Kbytes de RAM em um micro se só podemos acessar 32. Entretanto ao conectarmos um disk-driver ao MSX, ele precisa ter 64 Kbytes de RAM para poder carregar o sistema operacional do diskette. No capítulo 6 o uso do disk-drive es-está mais detalhado.

Veja novamente na figura 0.9 como estão selecionados inicialmente os slots que contêm a ROM e a RAM no MSX Expert. Em outros micros MSX, a posição inicial da RAM nos slots e páginas pode ser diferente (figura 0.10) como, por exemplo, no HOTBIT.

Quando ligamos o MSX , a primeira ação do Sistema Operacional é "achar" em qual slot foi colocada a memória RAM. Encontrando o slot na qual ela está,ele reserva uma parte dessa memória para poder usá-la como rascunho. Esta parte da memória RAM é a área das "Variáveis do Sistema" e mais adiante vamos falar sobre ela com mais detalhes.

No momento precisamos apenas saber que o Sistema Operacional do MSX ocupa aproximadamente 4 Kbytes da RAM para seu uso próprio, como uma área de trabalho.

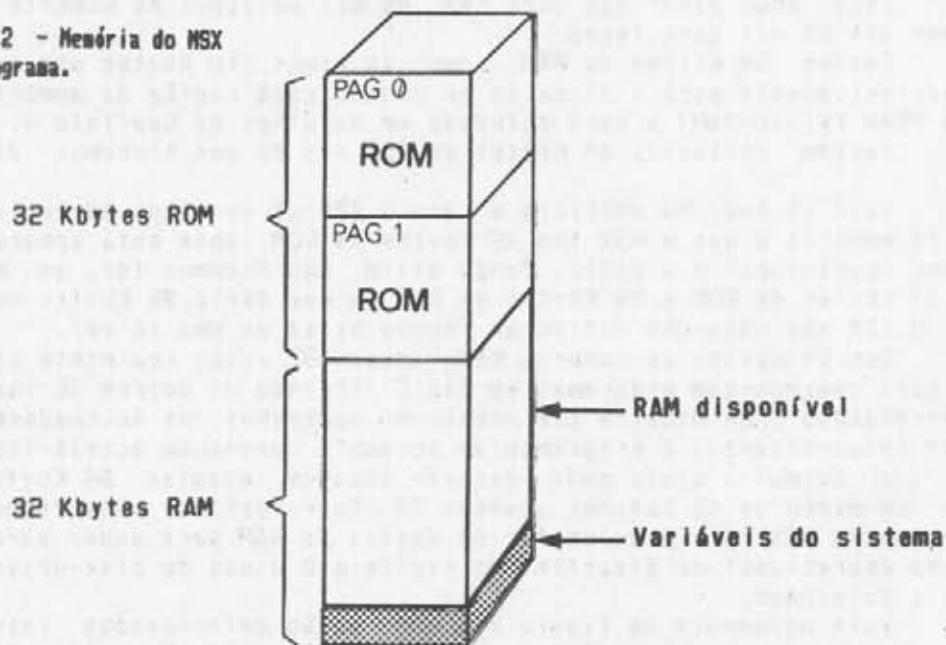
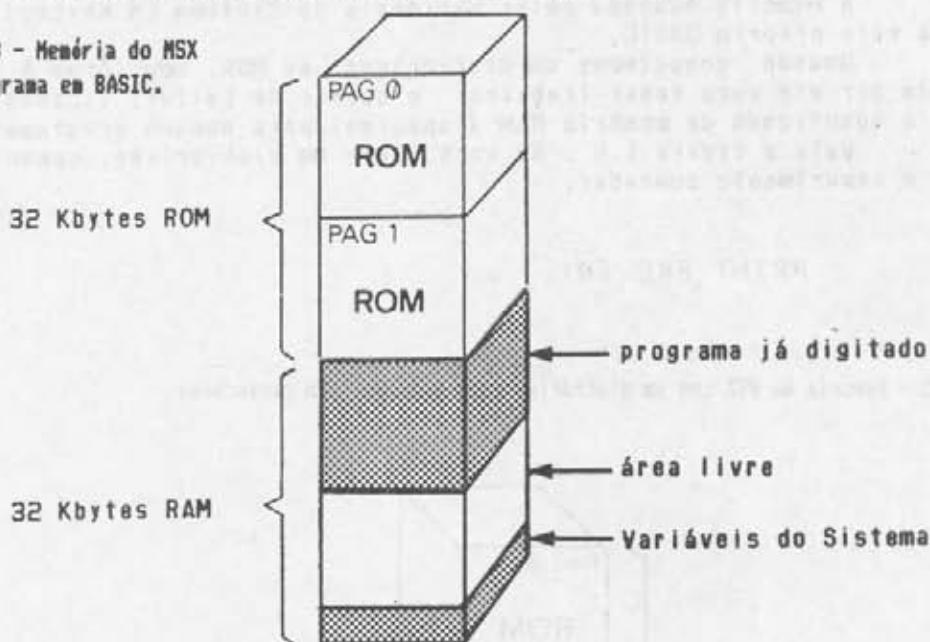
No final de tudo, após o MSX ter sido ligado, procurado e achado onde existe RAM, e reservada sua área de trabalho, ele nos indica o quanto "sobrou" de RAM para finalmente podermos usá-la!

Essa quantidade de RAM disponível é mostrada no vídeo logo que o micro é ligado (figura 1.1).

fig. 1.1 - Tela de apresentação dos MSX.

Veja na figura 1.2 como fica a memória do MSX.

fig. 1.2 - Memória do MSX sem programa.

Por isso, só temos 28 Kbytes livres para uso com BASIC. As variáveis do sistema utilizaram 4 Kbytes.

À medida em que vamos escrevendo um programa, este vai sendo colocado na memória RAM, diminuindo, assim, a quantidade disponível.

Após termos digitado uma certa quantidade de linhas de um programa, a memória RAM fica como mostra a figura 1.3.

fig. 1.3 - Memória do MSX com programa em BASIC.

A área de memória RAM ainda não ocupada (área livre) pode ser obtida facilmente através da função FRE (Ø).

Esta função, quando utilizada com um valor numérico entre parênteses (zero, por exemplo) nos devolve a quantidade que temos de RAM disponível para uso.

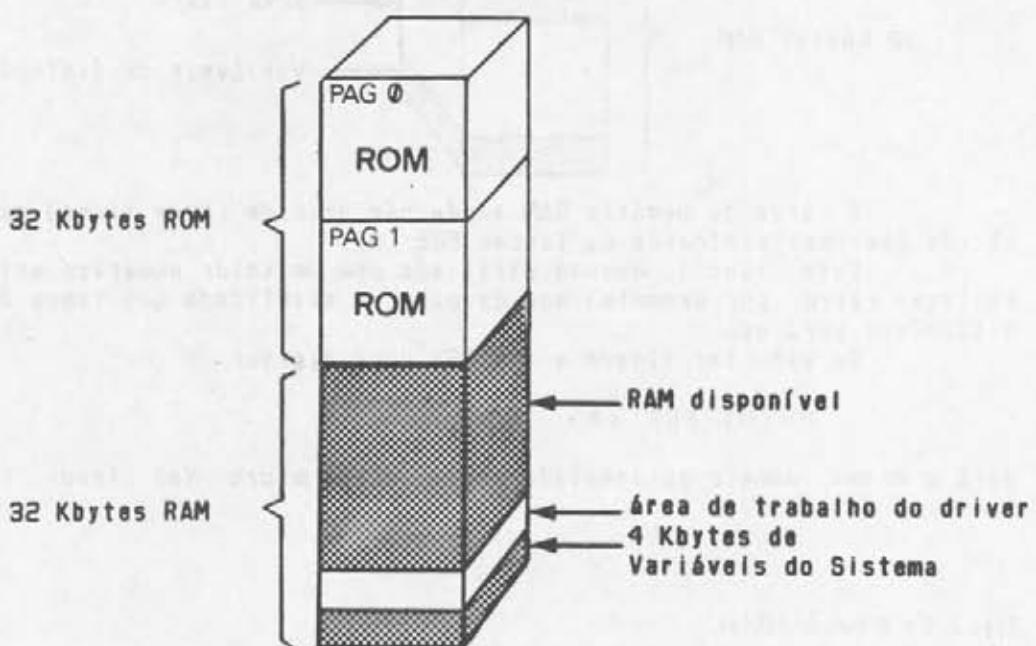
Se após ter ligado o seu MSX você digitar

```
PRINT FRE (Ø)
```

verá o mesmo número apresentado assim que o micro foi ligado (figura 1.4).

fig. 1.4 - A função FRE(n).

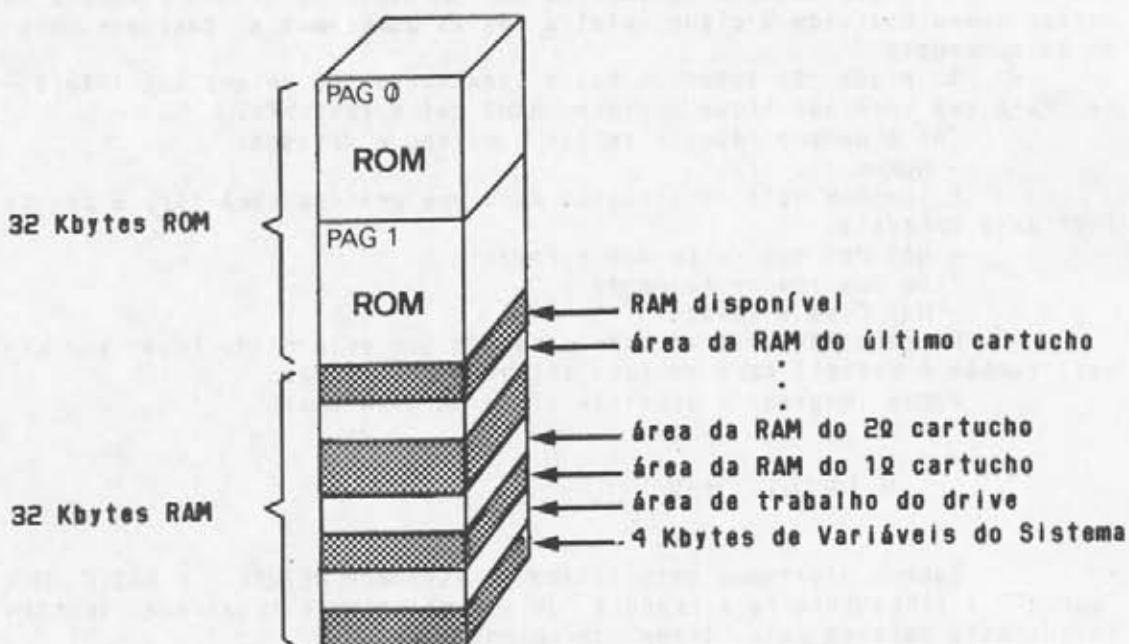
Portanto, sempre que você desejar saber quanto ainda lhe resta de memória livre, utilize a função FRE (0) e ela lhe dará a resposta.

Note que estes 28 Kbytes são disponíveis somente quando não estamos usando disk-drivers ou cartuchos!

A memória ocupada pelas Variáveis do Sistema (4 Kbytes) é utilizada pelo próprio BASIC.

Quando conectamos um disk-driver ao MSX, uma área à parte é alocada por ele para poder trabalhar e servir de buffer, ficando menor ainda a quantidade de memória RAM disponível para nossos programas!

Veja a figura 1.5 . Se você tiver um disk-driver, conecte-o ao micro e experimente comandar,

```
PRINT FRE (0)
```

fig. 1.5 - Memória do MSX com um disk-drive face simples, sem cartuchos.

```
MSX BASIC versão 1.0 Br
G r a d i e n t e
24455 bytes livres
Disk BASIC version 1.0
Ok
PRINT FRE(0)
 24455
Ok
■
```

Alguns cartuchos também precisam de área de memória RAM para seu próprio uso, e assim lá se vão mais alguns bytes!

No final de tudo, dependendo do número de cartuchos instalados e do disk-driver, teremos menos de 28 Kbytes para utilizar, ficando a RAM mais ou menos como mostra a figura 1.6 .

fig. 1.6 - Memória com disk-drive e cartuchos.

Como um programa é armazenado

Não fique muito desapontado ao saber que dos 80 Kbytes de RAM apenas 28 Kbytes (ou menos) sobraram para serem usados. Com o tempo você verá que, em geral, estes famigerados 28 Kbytes são mais do que suficientes mesmo para os programas e aplicativos mais complexos.

Para conseguir tal proeza, o BASIC tem que dar um jeito de "comprimir" os dados que lhe são fornecidos.

Ele consegue fazer isso através de tabelas que estão na ROM contendo todas as palavras reservadas da linguagem BASIC MSX e, a cada palavra, ele associa um código. Você verá isso no capítulo 2, onde serão mostradas todas as palavras reservadas e seus respectivos códigos (estes códigos são também conhecidos como "tokens").

Palavra reservada é uma sequência de caracteres que o micro reconhece como sendo especial e que ele pode entender. As palavras reservadas constituem o "vocabulário" do micro, e cada uma faz com que ele execute uma certa ação.

Palavras como PRINT, INPUT, IF, etc., são palavras especiais e fazem o BASIC MSX executar uma certa tarefa.

Para cada palavra reservada existe um código que a identifica sem haver a necessidade dele guardá-la inteiramente na memória.

Existe uma piada que ilustra muito bem o que acontece no micro. Ela é mais ou menos assim:

"Uma vez uma pessoa viu dois loucos no seguinte diálogo:

louco 1: - Vinte e dois.
louco 2: - Há Há Há - numa grande gargalhada.
louco 2: - Trinta.
louco 1: - Quá Quá Quá - rindo mais ainda.

Nisso, a pessoa que assistia à cena perguntou curiosa:

- Olha, por que quando um diz um número o outro cai na risada?

E um dos loucos responde:

- É que a gente já sabe de cor um monte de piadas e em vez de perder tempo contando a piada inteira nós as numeramos e contamos apenas os números!! "

A piada não acaba aí, mas é apenas essa parte que nos interessa. Para que você não fique curioso, aqui vai o restante:

"Aí a pessoa resolve testar o método e arrisca:

- Hummm ... 17!

E nenhum dois dois loucos move uma pestana para rir. A pessoa intrigada pergunta:

- Ué? Por que vocês não riram?!!

E um dos loucos responde:

- Há! Essa é velha!!"

O micro MSX utiliza algo parecido com esta piada (que, por sinal, também é velha!) para as suas palavras reservadas.

Vamos imaginar a seguinte linha de programa:

```
10 PRINT "MSX"
```

Quando digitamos esta linha e teclamos RETURN, o BASIC MSX "varre" a linha inteira à procura de alguma palavra reservada, substituindo esta palavra pelo "token" correspondente.

Veja na figura 1.7 como a linha foi introduzida na memória do micro e como a linha ficou depois de digitar-se RETURN.

Fig. 1.7 - Linha digitada e linha armazenada.

É como se a palavra PRINT fosse uma "piada".

Observe que a palavra PRINT ocupava 5 bytes (ela contém 5 letras) e foi transformada em apenas 1 byte, no qual foi colocado o "token" correspondente à palavra PRINT. Veja também que a tecla RETURN (↵) foi substituída por 0 para indicar que ali terminou uma linha. No final da brincadeira, uma linha digitada em 15 bytes foi armazenada em 12.

Além do texto em BASIC, temos também na linha o seu número, que o BASIC MSX consegue armazenar em dois bytes apenas.

A estrutura final de uma linha completa é a seguinte:

- Os primeiros dois bytes indicam o endereço de memória onde começa a próxima linha do programa
- Os próximos dois bytes indicam o número da linha
- Do 5º byte em diante está o texto em BASIC da linha, contendo os "tokens" em vez de conter todas as palavras (isto se a palavra for uma palavra reservada)
- O último byte de uma linha contém sempre 0, indicando o seu fim.

Veja como fica no micro o programa da figura 1.8 depois de ter sido digitado (se você não estiver familiarizado com notação hexadecimal, consulte antes o apêndice II).

fig. 1.8 - Programa na tela e na memória.

Na tela:

```
10 INPUT "NOME";A$
20 PRINT LEN(A$) : STOP
Ok
■
```

Na memória:

endereço &H 8001

11	80	0A	00	85	20	22	4E	4F	4D	45	22	3B	41	24	00
end. próx. linha		no. da linha		TOKEN do INPUT	ESPAÇO	"	N	O	M	E	"	;	A	$	fim da linha

endereço &H 8011

22	80	14	00	91	20	FF	92	28	41	24	29	20	3A	20	90	00
end. próx. linha		no. da linha		TOKEN do PRINT	ESPAÇO	TOKEN do LEN		(	A	$	)	ESPAÇO	:	ESPAÇO	TOKEN do STOP	fim da linha

endereço &H 8022

00	00	-

indica fim de programa

O programa, para ser digitado, precisou de 43 bytes (você precisa teclar 43 teclas para digitar o programa) e foram armazenados 33 bytes na memória do micro!!

Já deu para perceber que os 28 Kbytes de RAM são bem utilizados e conseguem armazenar programas de uma maneira bem racional.

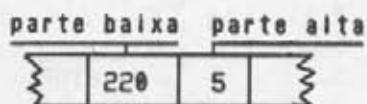
A essa altura você deve estar-se perguntando como conseguimos ver o programa direitinho quando usamos o comando LIST, se na memória do micro o programa está cheio de códigos!

É que, quando digitamos LIST, o micro faz exatamente o contrário do que quando digitamos RETURN ao final de uma linha. Ou seja, ele pesquisa nas tabelas da ROM os códigos das palavras reservadas (os "tokens") e mostra no vídeo a palavra completa. Assim, nós acabamos vendo o programa da mesma maneira que o digitamos, mesmo que na memória RAM estejam apenas os códigos ("tokens") de cada palavra reservada. No próximo capítulo você verá onde e como as palavras reservadas e suas respectivas "tokens" estão armazenadas.

Vamos ver um pouco mais detalhadamente esta forma de guardar as linhas de programas na RAM.

Quando digitamos o número da linha, o BASIC MSX o transforma (se for possível) em dois bytes, no seguinte formato: o 1o. byte contém a parte "baixa" do número e o 2o. byte contém a parte "alta" do número (veja o apêndice II).

A parte alta do número é conseguida utilizando a parte inteira da divisão do número da linha por 256, e a parte baixa é o resto (inteiro) desta divisão.

Vamos supor que o número de uma linha seja 1500. Quando digitamos uma linha de número 1500 o BASIC MSX divide este número por 256 e leva em conta apenas a parte inteira:

1500 ÷ 256 = 5.859375 ➡ 5 (parte inteira)

Depois disso, para conseguir a parte baixa do número ele toma o resultado obtido anteriormente (5) e multiplica por 256. Esse resultado é subtraído do número original e se obtém, assim, a parte baixa:

256 * 5 = 1280
1500 - 1280 = 220

Portanto, a linha de número 1500 é armazenada na memória assim:

fig. 1.9 - Número de linha na memória.

	parte baixa	parte alta	
...	220	5	...

É um processo um tanto complicado, mas tem suas vantagens. Utilizando este método é possível representar em apenas dois bytes qualquer número compreendido entre 0 e 65535, conseguindo assim uma boa economia de memória! É também por isso que a numeração das linhas pode ir de 0 a 65529.

Para recuperar o valor original do número da linha a partir

desses dois bytes, basta multiplicar a parte alta do número por 256 e somar ao resultado a parte baixa do número:

5 * 256 = 1280
1280 + 220 = 1500

Não podemos ter números de linhas superiores a 65535, pois não é possível armazená-lo em apenas dois bytes!

A mesma idéia é aplicada quando temos que representar um endereço de memória. O programa listado na figura 1.10 calcula um número em função do conteúdo de dois bytes ou esses conteúdos em função do número.

fig. 1.10 - Parte alta e parte baixa de um número.

```
10 INPUT "Qual o numero ";N
20 H=INT (N/256)
30 L=N-256*H
40 PRINT "Parte alta  (H)=";H
50 PRINT "Parte baixa (L)=";L
60 PRINT
70 INPUT "Agora entre a parte alta ";H
80 INPUT "E a parte baixa";L
90 PRINT "O numero e':";256*H+L
Ok
■
```

O Z80, como já vimos, consegue gerenciar até 65536 (ou 64 Kbytes) posições de memória (de 0 a 65535). Quando temos que guardar na memória um certo endereço,este é armazenado sempre em dois bytes: a parte baixa e a parte alta do número, do mesmo modo que são armazenados os números de linhas.

Neste livro você verá muitas vezes a sequência "parte baixa e parte alta" para representar um número e em muitas situações um endereço de memória será representado em dois bytes. Nos dois casos é usado o método que acabamos de descrever para representar um valor que não pode ser armazenado em um só byte. Quando se usam dois bytes, vem em primeiro lugar a parte baixa e em segundo lugar a parte alta.

Devido à origem "YANKEE" dos primeiros computadores, é muito comum encontrarmos os termos "High" e "Low" (alto e baixo, em inglês, respectivamente) e as letras "H" e "L" para representar a parte alta e parte baixa de um número.

Quando digitamos uma linha de um programa, nós primeiro digitamos o número da linha e depois os comandos que queremos que o computador execute naquela linha.

O número da linha serve para que o computador saiba o local exato do programa no qual queremos que a linha que está sendo digitada seja "encaixada".

Para que o BASIC localize na memória a posição exata onde a linha digitada deve ser inserida, ele coloca um indicador em cada linha para facilitar o trabalho de procura.

Esse indicador (ou ponteiro) é formado pelos dois bytes que estão antes do número da linha (que também está representada por dois bytes).

Apesar de você só ter digitado o número da linha e os respectivos comandos, o BASIC MSX colocou mais dois bytes para que ele possa localizar na memória RAM com muita rapidez cada linha em separado (veja novamente a figura 1.8).

Ao término de cada linha de programa existe um byte com valor igual a 0 para que o BASIC MSX saiba que ali terminou uma linha de programa e que logo em seguida deverá vir a próxima linha a ser executada.

Veja portanto que, para cada linha digitada são sempre gastos pelo menos 6 bytes:

- 2 para o BASIC poder localizar rapidamente o endereço da próxima linha na memória (epl).
- 2 para o número da linha no programa (nl).
- 1 para o comando que queremos que seja executado (cada linha do programa tem pelo menos um comando)
- 1 para indicar o término da linha (este byte é sempre igual a 0)

Compare os dois exemplos a seguir (figura 1.11). Veja que os programas executam a mesma função, porém um ocupou menos memória do que o outro.

fig. 1.11 - Multi-instrução e economia de memória.

```
10 INPUT A
20 PRINT A
30 END
```

8001

8009

8011

8017

```
10 INPUT A : PRINT A : END
```

8001

8013

Obs: os números estão em hexadecimal (veja apêndice II).

Repare que no primeiro programa foram gastos 24 bytes contra 20 bytes do segundo programa!

Ao elaborarmos um programa com várias instruções em uma mesma linha, estamos economizando memória. Em compensação ele fica um pouco mais difícil de se ler (para um humano, não para o micro!).

Fica a seu critério escolher a primeira ou a segunda maneira de se escrever um programa, optando pela economia de memória ou pela legibilidade, estética ou didática.

PEEK e POKE

Você já deve estar curioso para saber como "ver" o que está na memória do micro.

No BASIC MSX existe uma função que torna possível saber o que está armazenado em cada posição da memória, bastando para isso digitar:

```
PRINT PEEK (endereço)
```

Esta função recebe, como argumento, um valor (ou uma variável) correspondente a um endereço de memória.

Por ser um endereço de memória, o valor só pode estar entre 0 e 65535 (ou de -32768 a +32767). Se for usado um valor negativo como endereço, o BASIC MSX irá somar 65536 ao valor fornecido, para encontrar o endereço correto.

A função PEEK obtém o valor do byte que está armazenado no endereço que foi especificado como argumento. Portanto, a função PEEK só obtém valores entre 0 e 255.

Antes de usarmos a função PEEK, vamos ver um comando semelhante:

```
POKE endereço,byte
```

Este comando serve para inserir um byte num dado endereço de memória.

O endereço fornecido no comando POKE tem as mesmas restrições que na função PEEK, ou seja, deve ser um número entre 0 e 65535.

O byte também tem restrições: deve estar entre 0 e 255, já que estes são o menor e o maior valor que podemos representar em apenas um byte.

Ao usarmos o comando POKE, devemos tomar certos cuidados. Quando "pokeamos" um valor num endereço qualquer da RAM, o valor que estava naquele byte anteriormente é destruído, dando lugar ao novo valor que fornecemos.

Dependendo da posição de memória em que damos o POKE, podemos estar destruindo importantes informações, fazendo com que o BASIC fique "perdido" ou então "maluco"! Se isso ocorrer, não se preocupe, basta desligar e ligar novamente o computador que tudo volta ao normal, com exceção do seu programa que estava na RAM, que será apagado da memória.

Mais um cuidado deve ser tomado quanto ao comando POKE: só podemos utilizar o POKE em uma região da memória que possa ter seus bytes alterados, ou seja, na RAM!

Se você tentar dar um POKE a um endereço no qual haja memória ROM, o POKE não terá efeito! Ele será executado normalmente, mas como a memória ROM não consegue ser regravada, o POKE não terá efeito. Sendo assim, ao ver um programa que utiliza o comando POKE, repare que o endereço deste comando será sempre um valor superior a 32767, pois a memória RAM inicia a partir do endereço 32768 e vai até o endereço 65535.

O comando POKE geralmente é usado para introduzir um programa escrito em linguagem de máquina na memória do computador, ou para "tapearmos" o micro, fornecendo valores "malucos" em posições estratégicas.

Vamos ver, agora, alguns exemplos de utilização da função PEEK.

Na figura 1.12 temos um programa que mostra os dados na memória do computador.

fig. 1.12 - Programa DUMP 1.

```
10 INPUT "Endereco ";E
20 CLS
30 FOR I=1 TO 20
40 X$=HEX$(E)
50 X$=RIGHT$("0000"+X$,4)
60 PRINT X$;" - ";
70 X$=HEX$(PEEK(E))
80 X$=RIGHT$("00"+X$,2)
90 PRINTX$:E=E+1
100 NEXT I
110 PRINT
120 GOTO 10
Ok
█
```

Após tê-lo introduzido, rode-o digitando RUN e RETURN.

Ao ser executado, o programa pergunta qual o endereço da memória que você quer ver. Este endereço pode ser digitado em decimal ou em hexadecimal (nesse caso, é necessário digitar &H antes do endereço).

Como resposta a essa pergunta, digite:&H8001 (ou 32769) que é o endereço da RAM onde um programa é guardado.

Com isso, você está vendo como o programa que você acabou de digitar está armazenado na RAM!

Examine a listagem do programa e vá conferindo com o que aparece no vídeo; tente localizar cada "token", cada término de linha, etc.

Quanto ao programa em si não há nada de especial, mesmo assim vamos dar uma breve examinada nele.

Na linha 10 ele pede o endereço a partir do qual se quer examinar.

Da linha 30 a 100 está um laço para mostrar o conteúdo de 20 endereços.

A linha 40 coloca em X$ o endereço que está mostrado no formato hexadecimal, sendo "arrumado" na linha 50 para que contenha sempre 4 dígitos.

A parte "quente" do programa está na linha 70 , onde X$ recebe, no formato hexadecimal, o conteúdo do endereço indicado pela variável E.

Na linha 80 esse valor é "acertado" para conter sempre 2 dígitos e, depois, é mostrado na linha 90 . Nessa mesma linha o endereço é aumentado em uma unidade e, na linha 100, como já foi comentado, é fechado o laço.

Rode o programa várias vezes digitando endereços diferentes a cada rodada. Vasculhe bem a RAM e veja como há diferença entre aquilo que vemos normalmente no vídeo usando o comando LIST e o que realmente é armazenado na memória.

Na figura 1.13 temos o mesmo programa só que com uma pequena alteração: além de mostrar cada byte no formato hexadecimal ele mostra (se possível) o mesmo byte como um caractere.

Faça as alterações, rode o programa e repare como fica mais fácil de achar as "coisas" na RAM.

fig. 1.13 - Programa DUMP 2.

```
10 INPUT "Endereco ";E
20 CLS
30 FOR I=1 TO 20
40 X$=HEX$(E)
50 X$=RIGHT$("0000"+X$,4)
60 PRINT X$;" - ";
70 P=PEEK(E)
80 X$=HEX$(P)
90 X$=RIGHT$("00"+X$,2)
100 PRINT X$;" -- ";
110 IF P<32 OR P>127 THEN P=32
120 PRINT CHR$(P)
130 E=E+1
140 NEXT I
150 PRINT
160 GOTO 10
Ok
■
```

Com mais algumas alterações, você pode "embelezar" o programa. Digite o programa da fig. 1.14 e execute-o. Note que agora temos uma aparência mais bonita e muito mais informações apresentadas no vídeo. Se você quiser, pode gravá-lo em fita cassete para usá-lo mais tarde.

Quanto ao comando POKE, vamos ver apenas como utilizá-lo para carregar na RAM um programa em linguagem de máquina.

Normalmente os códigos em linguagem de máquina são colocados na forma hexadecimal nas linhas DATA.

Veja e estude o programa da fig. 1.15.

Este programa apenas carrega na RAM o programa em linguagem de máquina. Para executá-lo são necessários outros comandos e funções que serão descritos mais adiante.

Os códigos estão na linha 60. A linha 30 lê cada um deles e a linha 40 os coloca na região da RAM onde serão usados como um programa em linguagem de máquina.

fig. 1.14 - Programa DUMP.

```
10 INPUT "Endereco ";E
20 CLS
30 PRINT "END. -- HEXA --- ASCII"
40 PRINT
50 FOR E=E TO E+80 STEP 4
60 PRINT RIGHT$("0000"+HEX$(E),4);" - "
;
70 FOR I=0 TO 3
80 PRINT RIGHT$("00"+HEX$(PEEK(E+I)),2)
;" ";
90 NEXT I
100 FOR I=0 TO 3
110 P=PEEK(E+I)
120 IF P<32 OR P>127 THEN PRINT " ";ELS
E PRINT CHR$(P);
130 NEXT I
140 PRINT
150 NEXT E
160 GOTO 10
Ok
■
```

fig. 1.15 - Carregador de códigos.

```
10 CLEAR 200,&HBFFF
20 FOR E=&HC000 TO &HC003
30 READ X$
40 POKE E,VAL ("&H"+X$)
50 NEXT E
60 DATA CD,C3,00,C9
Ok
■
```

Variáveis do BASIC

Vamos dar uma breve examinada na estrutura dos dados armazenados na RAM para podermos explorar ao máximo os recursos oferecidos pelo Basic.

No MSX existem variáveis numéricas e variáveis não numéricas (variáveis "string"). Dentre as variáveis numéricas existem ainda as do tipo "inteira", "precisão simples" e "precisão dupla".

Cada tipo de variável tem seu uso mais indicado em uma dada situação do que em outra.

Para conseguirmos elaborar um programa que fique bem rápido na hora de ser executado, é bom saber como o Basic irá tratar cada variá-

vel, cada laço FOR...NEXT, etc. O uso adequado das variáveis dentro de um programa pode fazê-lo ficar menor, mais confiável e rápido, ao invés de deixá-lo ficar "horas" em simples cálculos numéricos!

Variáveis inteiras

As variáveis inteiras, como o próprio nome já diz, são variáveis que só conseguem armazenar números inteiros e que estejam na faixa de -32768 e até +32767 .

A primeira vista, pode parecer que este tipo de variável não tem muita aplicação mas, ao contrário, as variáveis inteiras devem ser usadas sempre que possível!

Este tipo de variável é armazenada na memória em apenas dois bytes. Nesses dois bytes, o mais significativo dos 16 bits presentes representa o sinal, e os 15 bits restantes representam o valor. Como cada bit pode conter 0 ou 1, com 15 bits conseguimos 2 elevado a 15 configurações diferentes, ou seja, 32768 possíveis combinações de zeros e uns. Por isso as variáveis inteiras só podem armazenar números entre -32768 e +32767.

Aparentemente, poderíamos ter o número 0 representado de duas formas: +0 e -0. Isto é desnecessário, afinal zero é zero. Porisso conseguimos representar desde -32678 até +32767 em vez de -32767 até +32767.

Não é por acaso que foram utilizados dois bytes. Apesar do Z80 (CPU do MSX) ser de 8 bits, ele tem capacidade de fazer somas e subtrações com 16 bits sem muita complicação e perda de tempo.

As operações de soma, subtração, multiplicação e divisão feitas entre variáveis inteiras são efetuadas utilizando-se comandos próprios do Assembly Z80 em vez de um complexo algorítmo de operações em ponto flutuante (como seria no caso da utilização de variáveis de precisão simples ou de precisão dupla).

Por esses motivos (pouco espaço ocupado na memória, 2 bytes, e grande velocidade de operação) é que devemos utilizar ao máximo as variáveis inteiras.

Na memória, estes dois bytes estão no mesmo formato visto anteriormente nos números das linhas: o primeiro byte contém a parte baixa do número (L) e o segundo contém a parte alta (H).

Para dizermos ao BASIC MSX que uma certa variável deve ser "tratada" como variável inteira, devemos colocar o símbolo de porcentagem (%) após o nome da variável ou, logo no início do programa, declararmos quais variáveis deverão ser assumidas como inteiras através do comando DEFINT.

Veja na figura 1.16 um exemplo de programa que utiliza variável inteira definida explicitamente:

fig. 1.16 - Variável inteira.

```
10 I%=5
20 PRINT I%,
30 I%=I%+1
40 GOTO 20
Ok
■
```

Sempre que o BASIC MSX encontrar o símbolo % após o nome de uma variável, ele a tratará como inteira.

Na figura 1.17 temos o mesmo programa, só que a variável I foi declarada inteira implicitamente através do comando DEFINT.

fig. 1.17 - Definição implícita.

```
10 DEFINT I
20 I=5
30 PRINT I,
40 I=I+1
50 GOTO 30
Ok
█
```

Note que, no exemplo anterior, todas as variáveis que começarem com a letra I (I1, IA, IX, etc.) serão consideradas inteiras!

Variáveis de precisão simples

Existem casos em que as variáveis inteiras não dão conta do recado.

Em algumas situações precisamos utilizar números decimais e não apenas números inteiros, ou mesmo números que não estejam entre -32768 e 32767. Nesses casos, temos que abrir mão da economia de memória e velocidade de processamento oferecidas pelas variáveis inteiras em pról da maior capacidade das variáveis de precisão simples.

Esse tipo de variável tem 6 dígitos de precisão, não importando em que lugar está a vírgula (ou melhor, o ponto) decimal. Por exemplo: 1.23456 é um número que é guardado com a mesma precisão que 768.510. Já o 132.4569, não consegue ser armazenado em uma variável de precisão simples corretamente, afinal ele tem 7 dígitos e não 6 !

Para poder visualizar melhor, experimente digitar os seguintes exemplos:

exemplo 1:
```
X!=123.456
 PRINT X!
```

resposta: `123.456`

exemplo 2:
```
X!=123.4569
 PRINT X!
```

resposta: `123.457`

Repare, no exemplo 2, que quando tentamos dar mais de 6 dígitos para a variável X , o BASIC MSX arredondou o número para 6 dígitos e isso é independente do lugar do ponto decimal!

O símbolo de exclamação (!) diz ao BASIC MSX que aquela variável (X) deve ser tratada como variável de precisão simples.

Outra maneira de dizer que queremos precisão simples para certas variáveis é através do comando DEFSNG.

Podemos utilizar o comando DEFSNG de maneira semelhante à usada nas variáveis inteiras.

Além dessa precisão (6 dígitos) existe ainda um expoente que pode ir de -64 a +63.

Em uma variável de precisão simples, portanto, nós podemos armazenar números desde $-9.99999 * 10^{63}$ até $+9.99999 * 10^{63}$, o que já satisfaz a maioria dos casos.

Para digitarmos o expoente de um número de precisão simples utilizamos a tecla E.

Veja alguns exemplos:

X! = 1.23E14 equivale a 1.23×10^{14}

A! = 2.5E-1 equivale a 2.5×10^{-1} ou 0.25

Y1! = -5.423E+2 equivale a -5.423×10^{2}

O BASIC utiliza o ponto decimal em vez de vírgula decimal porque foi "criado" (ou desenvolvido, se preferir) em um país de origem Anglo-Saxônica, e lá a notação é o ponto, diferente dos países Latinos, onde a notação é a vírgula.

Para o BASIC MSX poder armazenar valores das variáveis de precisão simples são utilizados 4 bytes de memória.

Desses 4 bytes, um é usado para o expoente do número e seu respectivo sinal, e os demais para o número propriamente dito.

Não vamos nos aprofundar muito no assunto de como são formados os números a partir desses quatro bytes, senão acabaríamos fugindo muito do objetivo deste livro.

Basta saber que o BASIC, ao fazer contas com números de precisão simples, usa um algoritmo muito complexo e acaba levando um bom tempo para executá-lo se comparado com o tempo despendido no tratamento de variáveis inteiras.

Lembre-se disso para utilizar variáveis em seus programas!

Variáveis de precisão dupla

Para as variáveis de precisão dupla, o BASIC MSX permite que até 14 dígitos sejam armazenados sem que haja perda de informação ou arredondamento.

É claro que para se conseguir isso gasta-se mais memória: 8 bytes.

Quanto ao expoente, continua igual ao das variáveis de precisão simples: de -64 a +63.

Para indicar ao BASIC MSX que alguma variável deve ser considerada de precisão dupla, coloca-se o símbolo # após o nome da variável, ou usa-se o comando DEFDBL no começo do programa. Para digitar o expoente, usa-se a letra D em vez da letra E. Veja alguns exemplos:

```
X#=1500
A#=1.5D-4
```
(equivale a 1.5×10^{-4})

É claro que para manipular números dessa ordem o Basic MSX ocupa muito mais tempo do que quando opera com variáveis de outro tipo. Por isso é bom evitar usar variáveis de precisão dupla, elas ocupam muita memória e fazem o MSX perder muito tempo nos cálculos. Elas devem ser usadas somente nos casos em que precisamos de mais de 6 algarismos significativos.

Mais algumas observações quanto às variáveis numéricas se fazem necessárias:

1) O número de bytes ocupados por cada tipo de variável não depende do valor dado à variável, mas sim do seu tipo.

Por exemplo:

X!=1 ocupa 4 bytes de memória
X!=1.5321E-20 também ocupa 4 bytes de memória

2) Quando indicamos explicitamente o tipo de variável que queremos usando %, ! ou #, este prevalece sobre o comando DEFxxx utilizado.

Por exemplo:

```
10 DEFINT A
20 AC=5
30 AI=10
40 AB!=50
50 AX#=3
```

Na linha 10 definiu-se que todas as variáveis qua comecam com a letra A serão inteiras, por isso, as variáveis AC e AI (linhas 20-30) serão variáveis inteiras, mas a variável AB! (linha 40) será de precisão simples e a variável AX# (linha 50) será de precisão dupla!

3) Ao ser ligado, o BASIC MSX assume que todas as variáveis serão de precisão dupla.

Só para termos uma idéia do tempo gasto nos cálculos com os diversos tipos de variáveis, digite o programa da figura 1.18 e rode-o com o comando RUN.

fig. 1.18 - Teste de velocidade.

```
10 TIME=0:FOR X%=1 TO 5000:NEXTX%
20 PRINT "Tempo X% =";TIME/60
30 TIME=0:FOR X!=1 TO 5000:NEXTX!
40 PRINT "Tempo X! =";TIME/60
50 TIME=0:FOR X#=1 TO 5000:NEXTX#
60 PRINT "Tempo X# =";TIME/60
Ok
█
```

Observando os resultados, vimos que o BASIC MSX demorou 4.03 segundos para fazer as 5000 contas com variáveis inteiras, 8.97 segundos para fazer as mesmas contas com a variável de precisão simples e 10.35 segundos com a de precisão dupla.

As vezes o próprio BASIC se encarrega de decidir quais, entre os números digitados, convém manter em precisão dupla, e quais com simples.

Variáveis strings

Para armazenar as variáveis do tipo string, o BASIC MSX usa 3 bytes que servem como indicadores dos dados da string, independentemente do seu tamanho.

Quando digitamos, por exemplo:

```
A$ = "ABACATE"
```

o BASIC MSX reserva na memória 3 bytes como indicadores (ou "pointers") para posteriormente saber o que há guardado na variável A$.

Esses 3 bytes são determinados da seguinte maneira: o primeiro byte indica o tamanho da string, e o segundo e terceiro bytes indicam

o endereço de memória onde realmente estão os dados, ou seja, a palavra "ABACATE".

Pelo fato de ter apenas um byte para indicar o tamanho da string, só podemos ter variáveis string com no máximo 255 caracteres (255 é o maior número que se pode ter em apenas 1 byte).

Se você tentar colocar mais de 255 caracteres em uma variável string, o Basic MSX lhe dará a seguinte mensagem de erro:

```
String too long
```

que quer dizer que a string é muito longa (no Hot-Bit a mensagem é em português).

Os outros 2 bytes contêm o endereço da memória RAM onde foi colocado o texto da variável. Esse endereço fica em um região de memória reservada especificamente para esta finalidade.

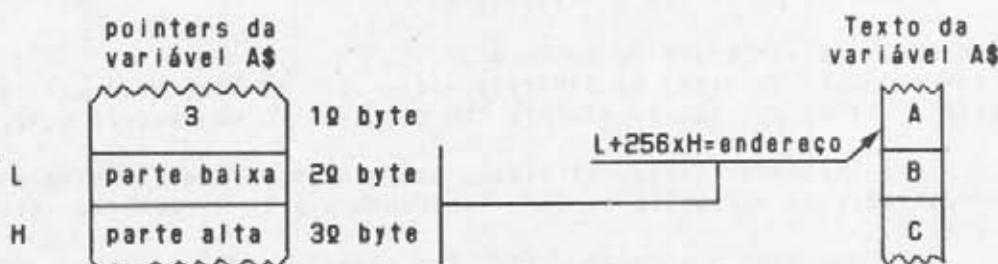
As variáveis strings podem mudar de tamanho durante a execução do programa, e para evitar o trabalho e a perda de tempo rearranjando espaço na memória, utiliza-se essa estrutura de bytes indicadores (ou "pointers"). Os "pointers" ficam numa posição fixa para cada variável, mas os conteúdos vão mudando de lugar.

A região da RAM onde são armazenados os textos das variáveis strings têm um certo tamanho, de modo que não podemos ir criando variáveis strings "a torto e a direito" sem estudar previamente quanta memória está disponível no programa.

O tamanho dessa área exclusivamente reservada para as variáveis strings é definido pelo comando CLEAR que veremos mais adiante.

Como se já não bastasse toda essa complicação, o BASIC MSX tem uma forma toda "peculiar" de trabalhar nessa área da RAM.

Vamos ver isso com o programa da figura 1.19.

fig. 1.19 - Strings.

```
10 A$="ABC"
20 A$=A$+"XYZ"
30 B$=A$
40 PRINT B$
Ok
```

Ao executar a linha 10, o BASIC MSX cria os 3 bytes para a variável A$ (3 bytes dos pointers).

No primeiro byte ele coloca o número 3 (3 é o tamanho da variável A$) e no segundo e terceiro bytes coloca o endereço da RAM onde as letras "ABC" estão armazenadas (veja figura 1.20).

fig. 1.20

Aí acontece algo interessante: o objetivo da linha 10 é criar uma variável A$ contendo as letras "ABC". O BASIC MSX deveria então reservar 3 bytes da RAM para os "pointers" necessários e depois "colocar" as letras "ABC" na área reservada para as variáveis string, acertando os a seguir os pointers para indicarem aquela região.

"Deveria", pois na verdade não é isso o que acontece! Para que copiar as letras "ABC" que já estão presentes na memória, na linha 10, para uma outra região da RAM? Afinal, se o que queremos é ter A$ com "ABC" e já existe "ABC" na própria linha do programa, é mais rápido e econômico fazer os pointers da variável A$ apontarem para a própria linha do programa do que ter que duplicar informações, gastando memória desnecessariamente!

Sendo assim, sempre que tivermos uma linha do tipo da linha 10 do programa anterior, o BASIC MSX nem "mexe" na área reservada para as variáveis string e utiliza os dados contidos na própria linha do programa, evitando duplicidade de dados e desperdício de memória!

Quando for ser executada a linha 20, o BASIC MSX "percebe" que vai haver alguma modificação nos dados na variável A$ e que não será mais possível que seus pointers continuem apontando para o meio da linha 10. Afinal, quando a linha 20 terminar de ser executada, a variável A$ terá "ABCXYZ" e não mais "ABC" apenas.

Aí sim, o BASIC MSX copia os dados da variável A$ para a área de texto das strings, acrescenta a elas as letras "XYZ" e apenas modifica os pointers de A$ para apontarem para esta área e não mais para o meio do programa!

Outro fato curioso ocorre quando a linha 30 é executada.

O que se deseja é ter B$ igual a A$.

O BASIC MSX cria então os 3 bytes de pointers para a variável B$ e simplesmente copia para eles os mesmos dados contidos nos pointers de A$. Note bem, não copia o texto, apenas os 3 bytes dos pointers!

Como resultado final nós temos a sequência "ABCXYZ" apenas uma vez na memória RAM, mas com duas variáveis (A$ e B$) como "donas" desses dados.

Todos esses detalhes do BASIC MSX existem apenas para um melhor aproveitamento da memória RAM, evitando desperdícios.

Tenha sempre isso em mente quando for utilizar variáveis string com comandos POKE ou sub-rotinas em Assembly, pois pode acontecer de você "pensar" que está alterando uma variável qualquer (através de POKE) e na verdade estar alterando muito mais coisas!

CLEAR

O comando CLEAR tem 3 finalidades:

1) Limpar todas as variáveis da memória;
2) Limitar o número de bytes da RAM reservados para as variáveis string;
3) Definir a última posição de memória RAM que o BASIC MSX poderá usar.

Este comando, quando utilizado sem nenhum parâmetro, simplesmente limpa todas as variáveis da RAM, liberando a área de memória ocupada por elas.

Se, logo após o comando CLEAR, for especificado um número, este irá indicar quantos bytes serão reservados para os textos das variá-

veis string. Por exemplo, digitando :

```
CLEAR 500
```

reservamos 500 bytes para as variáveis strings.

Se, logo após o número de bytes para as variáveis strings for especificado um segundo parâmetro, este indicará o último endereço de memória que o BASIC MSX poderá utilizar. Assim, se digitarmos:

```
CLEAR 300,&HE000
```

estaremos reservando 500 bytes para as variáveis strings e o BASIC MSX tem até o endereço &HE000 (em hexadecimal) disponível para o programa, variáveis, etc.

Sempre que formos utilizar um programa em Assembly (ou uma sub-rotina) temos que reservar uma certa quantidade de memória RAM onde ele será inserido através do comando POKE (ou através de BLOAD, se estiver gravado).

Se não fizermos essa reserva, corremos o risco do BASIC MSX estar utilizando a mesma área. Acabamos "atrapalhando" o BASIC (ou ele a nós!)

Com o comando CLEAR, podemos dizer ao BASIC MSX até qual endereço ele pode usar e que a área, após esse endereço,está sendo usada por um programa em Assembly. Assim, ninguém atrapalha ninguém.

Por isso, o comando CLEAR é normalmente um dos primeiros comandos encontrados em programas profissionais em BASIC, para limitar a área usada, reservar a área de RAM para as variáveis strings e limpar todas as variáveis até então utilizadas.

FRE

A função FRE (argumento) nos informa o quanto temos de memória RAM disponível para uso.

Dependendo do argumento que fornecemos, podemos ter a quantidade de RAM ainda livre para o nosso programa ou a quantidade de bytes livres na área de textos das variáveis string.

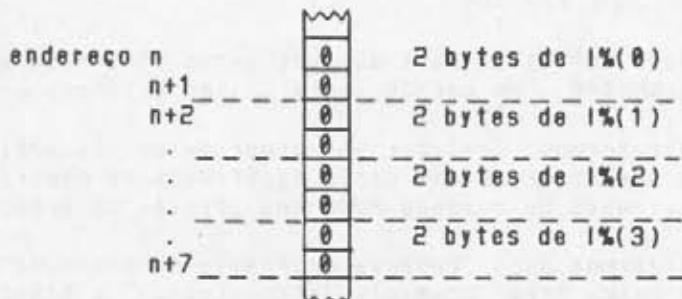
Se utilizarmos a função FRE com um argumento numérico (0, por exemplo), ela nos devolverá o quanto ainda temos de RAM para utilizarmos em um programa, como já vimos no capítulo 0.

Assim fica fácil saber o tamanho de um programa BASIC, subtraindo o valor fornecido pela função FRE (0) daquele que é mostrado assim que o micro é ligado (os bytes livres sem o programa na memória!).

Se o argumento da função FRE for não numérico ("", por exemplo), ela nos devolve quantos bytes estão livres na região reservada para as variáveis string.

Veja o exemplo a seguir:

fig 1.21

```
10 CLEAR 300
20 PRINT FRE("")
30 A$=STRING$(130,32)
40 PRINT FRE("")
Ok
█
```

Digite-o, rode-o e observe os valores mostrados no vídeo.

DIM

Ao definirmos uma matriz com o comando DIM, o BASIC MSX reserva na memória o espaço necessário para os elementos dessa matriz, um após o outro.

Se digitarmos DIM I%(3) a memória do micro fica da seguinte maneira:

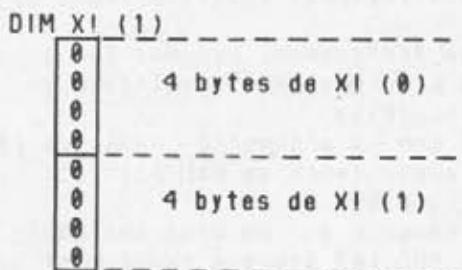
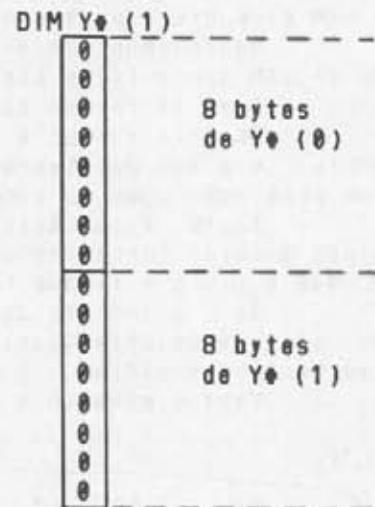



fig. 1.22

Ao usarmos o comando DIM, se especificarmos n como argumento de uma dimensão (DIM X(n), por exemplo), estaremos criando n+1 posições, pois o primeiro elemento tem índice 0, e não índice 1. Devido a isso, no exemplo anterior, foram dimensionadas 4 variáveis I% ao invés de 3.

A mesma estrutura continua válida para as variáveis de precisão simples e dupla (figura 1.23).

fig. 1.23

DIM para variável de precisão simples

DIM para variável de precisão dupla

Para as variáveis strings, o BASIC MSX dimensiona apenas os 3 bytes de pointers para cada elemento da matriz (figura 1.24).

fig. 1.24

0 0 0	Pointers de Z$ (0)
0 0 0	Pointers de Z$ (1)
0 0 0	Pointers de Z$ (2)
0 0 0	Pointers de Z$ (3)

Repare que, quando o comando DIM é executado, as variáveis numéricas são criadas com valores zerados e as variáveis string são criadas sem nenhum texto (comprimento igual a zero).

O comando DIM não é necessário quando se vai usar menos que 10 variáveis indexadas. Se num programa você utilizar, por exemplo, A(1), A(2), A(3) não é necessário dimensioná-las antes, pois o BASIC MSX, ao encontrar uma variável indexada que não tenha sido dimensionada, fa-lo-á automaticamente, reservando 11 elementos (de 0 a 10).

VARPTR

A função VARPTR é utilizada quando queremos saber em qual endereço da memória RAM está guardado o valor de uma variável.

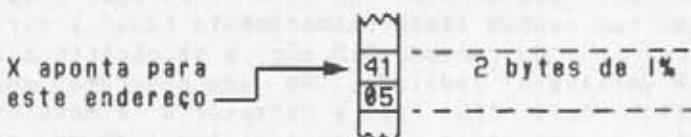
Ao atribuirmos um valor a uma variável, por exemplo, X=15, fazemos com que o BASIC armazene em algum lugar da memória RAM uma série de informações, de forma a permitir que, no decorrer do programa, possamos recuperar o valor dado a X (no caso, o valor 15).

A função VARPTR nos permite saber o local onde este "15" foi armazenado.

O simples fato de sabermos o local da memória do computador onde o dado está guardado não diz nada, é também preciso saber como o dado está armazenado, o que pode ser alterado e quais as vantagens (ou desvantagens) em se manipular tais informações!

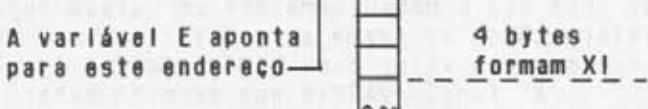
Os recursos que o BASIC MSX oferece no tratamento das variáveis são mais do que suficientes para as necessidades e situações encontradas ao se elaborar um programa. Mesmo com todos estes recursos disponíveis, porém, ainda nos deparamos com certas situações em que a linguagem BASIC MSX não dá conta do recado num tempo razoável. Sendo assim, temos que partir para o Assembly, onde a velocidade é a sua maior qualidade.

Os programas escritos em Assembly são mais complicados e trabalhosos para serem elaborados, mas são também mais velozes na hora da execução. É por este motivo, a velocidade, que são usadas algumas subrotinas escritas em Assembly (ou linguagem de máquina) junto com programas escritos em Basic!

Para que haja um entrosamento entre o programa escrito em BASIC e o escrito em Assembly, é necessário que este saiba onde localizar as variáveis que foram definidas anteriormente no programa BASIC.

Um programa (ou sub-rotina) em Assembly nunca deve criar uma variável BASIC, pois o BASIC MSX é um tanto "complicado" quanto às suas variáveis. É muito mais fácil deixá-lo criar as variáveis e passá-las para o Assembly do que fazer o Assembly criá-las.

Você leu, nos itens anteriores, como o BASIC MSX guarda os diversos tipos de variáveis na memória RAM. Vamos ver agora como usar a função VARPTR para saber o endereço da memória onde se localizam as variáveis.

Quando usamos VARPTR com uma variável inteira, o endereço que VARPTR devolve é o do primeiro byte dos dois que formam a variável (veja a figura 1.25).

fig. 1.25

```
10 I%=1345
20 X=VARPTR(I%)
```

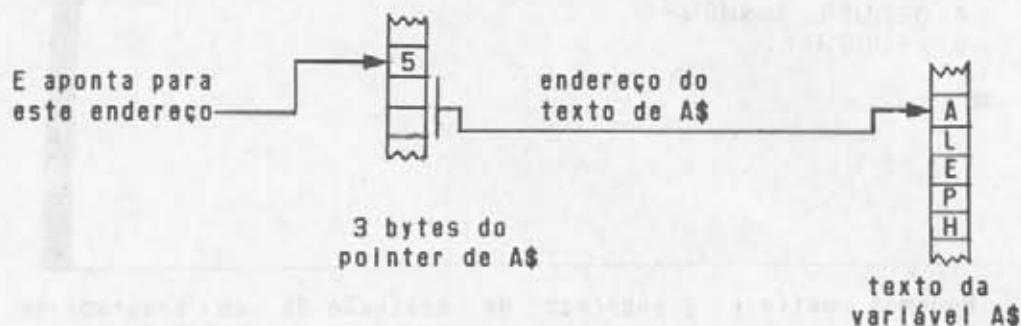
Se usarmos uma variável de precisão simples como argumento da função VARPTR, o endereço do primeiro dos 4 bytes que a constituem é devolvido pela função (figura 1.26).

fig. 1.26

```
10 X!=14
20 E=VARPTR(X!)
```

No caso do argumento ser uma variável de precisão dupla, o endereço do primeiro dos 8 bytes que o formam é devolvido (figura 1.27).

fig. 1.27

```
10 BA#=1986
20 K=VARPTR(BA#)
```

Atenção maior deve ser dada quando utilizamos uma variável string como argumento.

O endereço devolvido por VARPTR, quando o argumento desta função é uma variável string, é o primeiro dos 3 bytes que formam os pointers da variável em questão (veja a figura 1.28).

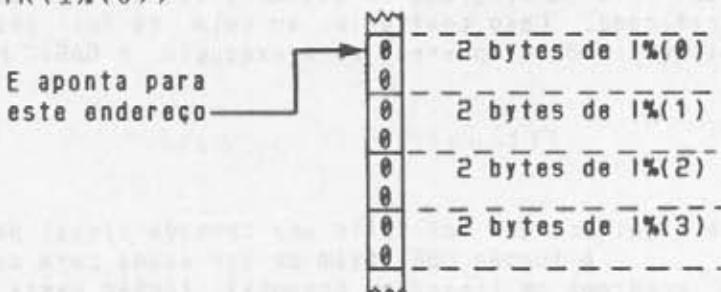
fig. 1.28

Também pode-se usar VARPTR com variáveis indexadas. Neste caso, o argumento da função VARPTR geralmente é o elemento 0 da matriz (veja figura 1.29).

fig. 1.29

```
10 DIM I%(3)
20 E=VARPTR(I%(0))
```

E aponta para este endereço

0 0	2 bytes de I%(0)
0 0	2 bytes de I%(1)
0 0	2 bytes de I%(2)
0 0	2 bytes de I%(3)

DEFUSR

O comando DEFUSR serve para definirmos o endereço de execução de um programa escrito em Assembly. Depois do programa ter sido carregado na memória, é preciso informar ao BASIC o endereço de entrada do programa em Assembly para que ele possa ser executado quando necessário.

Podemos utilizar o comando POKE para carregar o programa em Assembly ou carregá-lo da fita com o comando BLOAD.

Note que, ao definirmos um endereço de entrada de um programa (ou sub-rotina) em Assembly, não significa que ele exista realmente nesse endereço! Utilizando o comando DEFUSR estamos apenas definindo o endereço de entrada e nada mais!

Só podemos definir 10 endereços de execução de programas em Assembly com este comando (de 0 a 9). Se precisarmos de mais de 10, temos que redefinir uma delas antes de chamar a função USR. Por exemplo:

fig. 1.30 -

Podemos definir o endereço de execução de um programa em Assembly quantas vezes forem necessárias em um programa em BASIC.

O endereço de memória que será definido como entrada de um programa em Assembly pode ser qualquer um entre 0 e 65535. Assim, podemos utilizar várias sub-rotinas já prontas que estão na ROM ou criarmos nossas próprias sub-rotinas e colocá-las na RAM para depois serem executadas.

USR

A função USRn tem como finalidade dar início à execução de um programa escrito em Assembly, a partir do endereço previamente definido no comando DEFUSR.

Ao utilizarmos a função USR, o BASIC MSX verifica se foi definido um endereço de execução para esta função. Se tal endereço foi definido, então o programa em Assembly passa a ser executado no endereço especificado. Caso contrário, ou seja, se foi chamada a função USR e não foi definido o endereço para execução, o BASIC MSX acusa o seguinte erro :

```
Illegal function call
```

que significa que foi feita uma chamada ilegal para uma função.

A função USR, além de ser usada para dar início à execução de um programa em linguagem Assembly, também passa para ele valores de uma variável BASIC e depois disso retorna ao BASIC o valor obtido pelo programa em Assembly.

A rigor, a função USR não passa o endereço da variável, mas sim copia seu valor em um lugar fixo da memória. Depois que o programa em Assembly terminou de ser executado, a função USR copia novamente para a variável o valor contido no endereço citado.

Vamos supor a seguinte linha de programa:

```
10 K=USR3(Y)
```

Ao encontrar uma linha de programa com essas características, o BASIC MSX procede da seguinte forma:

1) O valor da variável Y é copiado de sua posição original para uma outra área da RAM situada a partir do endereço &HF7F6.

2) É, então, executado o programa em Assembly (a partir de um endereço na memória definido por um comando DEFUSR).

3) Ao terminar a execução do programa em Assembly , o BASIC copia os dados que estão a partir do endereço &HF7F6 e os atribui à variável K.

Além dessa movimentação toda, o BASIC MSX coloca no endereço &HF663 um valor que serve de "indicador" para o programa Assembly.

Esse indicador varia de acordo com o tipo de variável que será passada de um programa para outro, como segue:

- Para variáveis inteiras, o endereço &HF663 contém o valor 2, e o valor da variável é copiado em &HF7F8 e &HF7F9.

- Para variáveis de precisão simples, o endereço &HF663 contém o valor 4 e o valor da variável é copiado desde &HF7F6 até &HF7F9.

- Para variáveis de precisão dupla, o endereço &HF663 contém o valor 8 e o valor da variável é copiado desde &HF7F6 até &HF7FD.

- Para variáveis string, o endereço &HF663 contém o valor 3 , e os dois bytes que apontam para o texto da string são copiados em &HF7F8 e &HF7F9 contém o endereço dos 3 bytes da variável string.

Após um programa em Assembly ter terminado a execução, o BASIC MSX faz o inverso, movendo os dados a partir de &HF7F6 para a variável correspondente, de acordo com o valor presente em &HF663, valendo todas as regras que acabamos de descrever.

Portanto, quando utilizamos uma sub-rotina em Assembly que precisa de apenas uma (ou nenhuma) variável do BASIC, podemos usar USR para passar o valor de um programa para o outro sem dificuldade.

Quando, porém, temos mais de uma variável para passarmos de um programa BASIC para um programa Assembly e vice-versa, temos que utilizar uma matriz e a função VARPTR.

Primeiro deve ser dimensionada uma matriz através do comando DIM. Depois, coloca-se nessa matriz os dados a serem passados para o programa em Assembly.

Cria-se, então, uma variável inteira contendo o endereço do primeiro elemento dessa matriz através da função VARPTR.

Executa-se o programa em Assembly passando o endereço do primeiro elemento da matriz com a função USR.

Nesse ponto, o programa em Assembly, recebendo o primeiro elemento da matriz, pode facilmente manipular os demais elementos, já que eles estão um após o outro, como vimos anteriormente.

Vamos ver um exemplo de programa em Assembly para assentar bem todas as idéias aqui apresentadas.

Utilizaremos uma sub-rotina que recebe uma variável transformando todas as letras minúsculas, permitindo assim uma digitação sem a preocupação de saber se a tecla CAPS LOCK está ou não apertada.

Digite com atenção o programa listado na figura 1.31.O programa em Assembly está definido nas linhas DATA (linhas de 1050 a 1090) e sua listagem disassemblada pode ser vista na figura 1.32. Para maiores detalhes sobre o Assembly, veja o apêndice III.

fig. 1.31 - Programa em linhas DATA.

```
1000 CLEAR 300,&HBFFF
1010 FOR I=&HC000 TO &HC021
1020 READ X$
1030 POKE I,VAL("&H"+X$)
1040 NEXT I
1050 DATA 3A,63,F6,FE,03,C0,2A,F8
1060 DATA F7,7E,B7,C8,47,23,5E,23
1070 DATA 66,6B,7E,FE,61,38,07,FE
1080 DATA 7B,30,03,D6,20,77,23,10
1090 DATA F1,C9
1100 DEFUSR0=&HC000
1110 INPUT X$
1120 PRINT USR0(X$)
1130 GOTO 1110
Ok
█
```

Após digitar o programa em BASIC, rode-o e introduza vários caracteres maiúsculos e minúsculos, gráficos, etc., e verifique o resultado impresso, após a sub-rotina em Assembly ter sido executada.

fig. 1.32 - Programa disassemblado.

```
C000 3A63F6   LD   A,(F663H)
C003 FE03     CP   03H
C005 C0       RET  NZ
C006 2AF8F7   LD   HL,(F7F8H)
C009 7E       LD   A,(HL)
C00A B7       OR   A
C00B C8       RET  Z
C00C 47       LD   B,A
C00D 23       INC  HL
C00E 5E       LD   E,(HL)
C00F 23       INC  HL
C010 66       LD   H,(HL)
C011 6B       LD   L,E
C012 7E       LD   A,(HL)
C013 FE61     CP   61H
C015 3807     JR   C,C01EH
C017 FE7B     CP   7BH
C019 3003     JR   NC,C01EH
C01B D620     SUB  20H
C01D 77       LD   (HL),A
C01E 23       INC  HL
C01F 10F1     DJNZ C012H
C021 C9       RET
```

Variáveis do Sistema

A "área das variáveis do sistema" é a região da memória RAM na qual o BASIC MSX mantém informações a respeito de seu estado atual de funcionamento.

É nesta região da memória que temos todos os parâmetros mutáveis necessários para o perfeito funcionamento do micro.

Nos micros da linha MSX, esta área está localizada a partir do endereço &HF380 .

Através das variáveis do sistema é possível saber, por exemplo, se a tecla CAPS LOCK foi ou não pressionada, fazendo uma simples consulta a essa área da memória.

Vamos ver apenas algumas das muitas existentes, pois grande parte dessas variáveis são apenas "áreas de rascunho" do BASIC e não vamos conseguir nada de útil com PEEKs e POKEs nessa região.

Vamos dar um nome para cada variável apresentada. Este nome não tem nada a ver com os nomes que damos às variáveis em um programa BASIC e servirão apenas para facilitar o nosso trabalho quando, mais adiante, precisarmos nos referenciar a elas.

F380	RDPRIM	Pequena sub-rotina em Assembly para haver leitura entre os slots primários. O BASIC utiliza esta sub-rotina quando é executado o comando CALL.
F385	WRPRIM	Pequena sub-rotina em Assembly para haver gravação entre os slots primários.
F39A	USRTAB	Tabela dos endereços de execução dos programas em Assembly. F39A-F39B Contém o endereço de execução USR 0. F39C-F39D Contém o endereço de execução USR 1, e assim por diante.
F3AE	LINL40	Valor do último comando WIDTH no SCREEN 0.
F3AF	LINL32	Valor do último comando WIDTH no SCREEN 1.
F3B0	LINLEN	valor do último comando WIDTH no SCREEN 0. ou no SCREEN 1.
F3B1	CRTCNT	Número de linhas do vídeo que o BASIC MSX pode usar. Experimente dar um POKE &HF3B1,14 e utilizar o vídeo depois.
F3B2	CLMLST	A seguir vem 4 tabelas para o VDP usar. São os mesmos dados fornecidos pela função BASE (n).
F3B3	TXTNAM	Tabelas para SCREEN 0 .
F3B5	TXTCOL	
F3B7	TXTCGP	
F3B9	TXTATR	
F3BB	TXTPAT	

RAM - MEMÓRIA PARA USUÁRIO

Endereço	Nome	Descrição
F3BD	TXTNAM	Tabelas para SCREEN 1 .
F3BF	TXTCOL	
F3C1	TXTCGP	
F3C3	TXTATR	
F3C5	TXTPAT	
F3C7	TXTNAM	Tabelas para SCREEN 2 .
F3C9	TXTCOL	
F3CB	TXTCGP	
F3CD	TXTATR	
F3CF	TXTPAT	
F3D1	TXTNAM	Tabelas para SCREEN 3 .
F3D3	TXTCOL	
F3D5	TXTCGP	
F3D7	TXTATR	
F3D9	TXTPAT	
F3DB	CLIKSW	Indicador do "click" do teclado : = 0 não faz "click" ≠ 0 faz "click" Experimente dar alguns POKEs com 0 e 1 e aperte qualquer tecla para ver o resultado. Os 2 endereços a seguir indicam a posição do cursor:
F3DC	CSRY	Posição Y do cursor + 1 ;
F3DD	CSRX	Posição X do cursor + 1 .
F3DE	CNSDFG	Indicador de "mostra" ou "não mostra" as teclas de função no vídeo: = 0 não mostra as teclas; ≠ 0 mostra as teclas.
F3E8	TRGFLG	Indicador dos botões de tiro do joystick. Um bit para cada botão de tiro.

Experimente digitar e rodar o seguinte programa. Quando ele estiver rodando, aperte a barra de espaços e os botões de tiros dos joysticks.

```
10 X$=BIN$(PEEK(&HF3E8))
20 X$=RIGHT$("00000000"+X$,8)
30 PRINT X$
40 GOTO 10
```

F3E9	FORCLR	Código de cor dos caracteres.
F3EA	BAKCLR	Código de cor de fundo.
F3EB	BDRCLR	Código de cor da borda.
F414	ERRFLG	Código de erro (equivalente a ERR).
F415	LPTPOS	Valor do LPOS.
F663	VALTYP	Indicador do tipo de variável usado por USR.
F6AA	AUTFLG	= 0 se o comando AUTO não está em uso.
F6AB	AUTLIN	Número da linha sendo inserida pelo AUTO.
F6AD	AUTINC	Incremento para AUTO.
F6B3	ERRLIN	Número da linha onde ocorreu o erro (ERL).
F6B5	DOT	Número da linha para LIST.
F6B9	ONELIN	Número da linha para ON ERROR GOTO.
F6BB	ONEFLG	= 0 se ON ERROR GOTO não existe no programa.
F6BE	OLDLIN	Número da linha executada quando o programa encontra END ou foi teclado CONTROL + C.
F6C0	OLDTXT	Pointer para o próximo comando a ser executado.
F6C2	VARTAB	Pointer para a área de variáveis.
F6C4	ARYTAB	Pointer para a área de arrays.
F6C6	STREND	Fim da área de armazenamento das variáveis do BASIC.
F6C8	DATPTR	Pointer para os comandos DATA.
F6CA	DEFTBL	Tabela de 26 elementos indicando qual o tipo de variável a ser assumida ao se encontrar a primeira letra do nome de uma variável dentro de um programa BASIC.
F7C4	TRCFLG	Indicador do TRACE = 0 se não está sendo usada a função TRON.
F7C5	FBUFFR	Área usada pelas rotinas de cálculos matemáticos.
F857	RNDX	Valor do último RND gerado.
F85F	MAXFIL	Valor do MAXFILE.

F860	FILTAB	Endereço dos dados do arquivo.
F87D	SAVEND	Ultima posição de memória a ser gravada pelo comando BSAVE.
F87F	FNKSTR	Area dos textos das teclas de função. Você pode usar o comando POKE nesta área para mudar os textos das teclas de função.
FC48	BOTTOM	Endereço inicial da memória RAM a partir do qual o BASIC irá armazenar o programa (geralmente = &H8000).
FC4A	HIMEM	Endereço final da memória RAM disponível para o BASIC.
FCA8	INSFLG	Indicação de inserção ligada ou desligada.
FCA9	CSRSW	Indicador de "mostra" ou "não-mostra" o cursor.
FCAB	CAPST	Indicador da tecla CAPS LOCK ativada ou ou não-ativada.
FCAF	SCRMOD	Número do SCREEN em uso.
FCBF	SAVENT	Endereço inicial do comando BSAVE.

Capítulo 2

ROM - SISTEMA OPERACIONAL DO MSX

O que há na ROM

Quando um microcomputador é ligado, ele realiza automaticamente um grande número de operações antes de tranferir o controle ao usuário. Mesmo depois, ele é capaz de entender e executar quase duzentas instruções e ainda checar constantemente suas entradas e saídas.

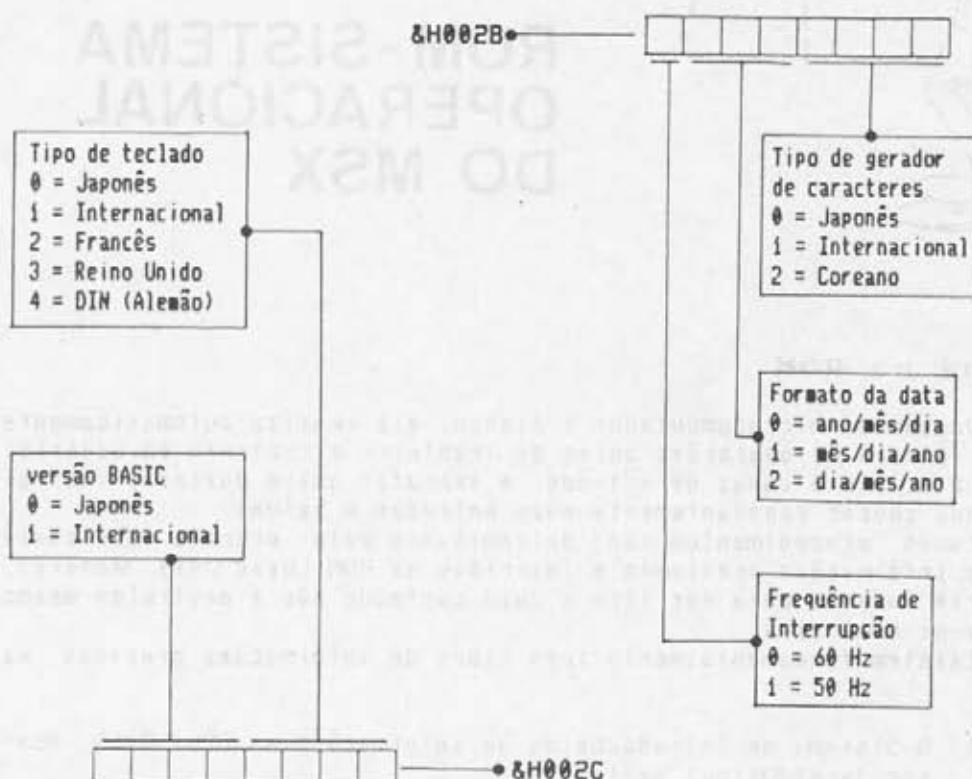
Esses procedimentos são determinados pelo próprio fabricante através de informações previamente inseridas na ROM (Read Only Memory), uma memória apenas para ser lida e cujo conteúdo não é destruído mesmo desligando-se a máquina.

Existem fundamentalmente três tipos de informações gravadas na ROM:

1) O Sistema de Entrada/Saída de informações na CPU: BIOS (Basic Input/Output System);
2) As tabelas: de caracteres, das palavras reservadas e suas tokens, das mensagens de erro e de identificação do fabricante.
3) O Interpretador BASIC, que é o programa responsável pela tradução do BASIC para a linguagem de máquina do Z80A.

As rotinas do BIOS são usadas sempre que ocorre troca de informações entre a CPU e os periféricos. São essas rotinas que controlam a PPI, o VDP, o PSG e a impressora, gerenciando todas as entradas e saídas de informação para ou da CPU.

As tabelas são necessárias para que o micro tenha algum "vocabulário". Ele necessita de caracteres para escrever de forma compreen-

sível as informações e precisa das palavras reservadas e suas tokens para identificar o BASIC. As mensagens de erro e a identificação do fabricante também têm que estar armazenadas para serem mostradas quando necessário.

O Interpretador BASIC é essencial para que o BASIC seja traduzido para linguagem de máquina do Z80A.

A seguir, veremos onde e como parte dessas informações estão armazenadas. Antes disso, porém, aprenderemos a identificar o sub-padrão de um micro MSX. Apesar de ser internacional, o padrão MSX permite pequenas variações na ocupação da ROM, na tabela de caracteres, na concepção do teclado, etc...

Existem alguns bytes no início da ROM que servem para identificar o sub-padrão adotado numa determinada máquina. Esses bytes são chamados ID bytes (IDentification bytes) e estão nos endereços &H002B e &H002C. Na figura 2.1 está esquematizada a organização das informações nesses bytes.

fig. 2.1 - IDentification Bytes.

A tabela de caracteres pode ser móvel, entretanto o endereço em que ela inicia é sempre armazenado nos bytes &H0004 e &H0005 (na forma L/H). Para obter o endereço inicial (em hexadecimal) da tabela basta comandar:

```
PRINT HEX$(PEEK(5));HEX$(PEEK(4))
```

As portas de comunicação com o VDP também podem variar de uma máquina para outra. Os endereços dessas portas estão armazenados nos bytes &H0006 e &H0007.

No Hot-Bit e no Expert, os valores desses 6 bytes são:

```
&H0004 ---- &H00BF
&H0005 ---- &H001B
&H0006 ---- &H0098
&H0007 ---- &H0098
&H002B ---- &B00010001
&H002C ---- &B00010001
```

BIOS (Basic Input/Output Sistem)

As rotinas do BIOS estão concentradas na primeira página da ROM (de &H0000 a &HBFFF). A seguir, algumas dessas rotinas serão comentadas e explicadas com exemplos. Como algumas rotinas necessitam dos dados contidos em certos registros do Z80, às vezes o programa exemplo poderá conter pequenas rotinas em linguagem de máquina. No apêndice III você encontrará uma breve introdução à programação em linguagem de máquina e ao ASSEMBLY Z80. Se você precisar de outras rotinas do BIOS, poderá encontrá-las de forma resumida no apêndice IV.

BEEP Essa rotina é chamada pelo endereço &H00C0 e não necessita de nenhum dado em especial. Basta chamá-la através do BASIC como mostra este simples programa.

fig. 2.2 - Programa BEEP.

```
10 DEFUSR=&HC0
20 A$=INPUT$(1)
30 A=USR(1)
40 GOTO20
Ok
```

O programa gera um beep toda vez que uma tecla é pressionada. O ponto de entrada da sub-rotina é definido pela instrução DEFUSR e o beep é acionado pela instrução A=USR(1), que executa a sub-rotina em linguagem de máquina contida no BIOS.

CLS A rotina CLS tem início no endereço &H00C3 e limpa a tela de vídeo toda vez que é chamada, não importando a SCREEN na qual se esteja trabalhando. Ela não necessita de nenhum tipo de dado nos registradores.

fig. 2.3 - Programa CLS.

```
10 DEFUSR=&HC3
20 '---------- SCREEN 0 ----------
30 SCREEN0
40 COLOR1,8
50 WIDTH32
60 LOCATE0,10
70 PRINT"AO FIM DA CONTAGEM VOU LIMPAR
A SCREEN 0 ATRAVES DA ROTINA CLS
     ( &H00C3 )"
80 FORL=500TO0STEP-1
90 LOCATE12,15:PRINTL
100 NEXT
110 '------ LIMPA TELA ----------
```

```
120 A=USR(1)
130 '--------- SCREEN 2 ---------
140 OPEN"GRP:"FOR OUTPUT AS #1
150 GOSUB420
160 COLOR 10,4,4:SCREEN2
170 FOR R=1TO50
180 CIRCLE(255*RND(TIME),192*RND(TIME))
,30*RND(TIME)
190 NEXT
200 LINE(0,0)-(255,20),4,BF
210 COLOR1 : PRESET(0,0) :PRINT#1,"VOU
LIMPAR A TELA GRAFICA DE ALTA RESOLUCAO
"
220 CLOSE#1
230 GOSUB420
240 '------ LIMPA TELA ----------
250 A=USR(1)
260 GOSUB420
270 '--------- SCREEN 3 ---------
280 COLOR1,7,7
290 SCREEN3
300 OPEN"GRP:"FOR OUTPUT AS #1
```

```
310 FOR R=3 TO9
320 CIRCLE(128,96),10*R,10
330 NEXT
340 PRESET(55,80):PRINT#1,"ALEPH"
350 PRESET(0,0):PRINT#1,"SCREEN 3"
360 GOSUB420:GOSUB420
370 '------ LIMPA TELA ----------
380 A=USR(1)
390 GOSUB420
400 COLOR 15,1,1:END
410 '----- PERDA DE TEMPO -------
420 FORM =0 TO 1000
430 NEXT
440 RETURN
Ok
■
```

INITFNK Essa rotina restabelece as Function Keys iniciais. A INITFNK tem início no endereço &H003E e a sua chamada pode ser feita diretamente pelo BASIC pois ela não necessita de nenhum dado especial contido nos registradores.

fig. 2.4 - Programa INITFNK.

```
100 CLS
110 PRINT"Pressione RETURN para alterar
 as       Fuctions keys"
120 IFINKEY$="" THEN 120
130 FOR R=1 TO 10
140 KEYR,"Aleph"
150 NEXT
160 PRINT"Pressione RETURN para restabe
lecer as  Fuctions Keys"
170 IFINKEY$="" THEN 170
180 DEFUSR=&H3E
190 A=USR(0)
200 KEY ON
Ok
■
```

DISSCR O acesso à rotina é feito pelo endereço &H0041 e quando executada ela desabilita o VDP, tornando a tela escura. Pode ser usada para escurecer a tela enquanto desenhos são elaborados pelo computador.

ENASCR Essa rotina é acessada pelo endereço &H0044 e habilita o VDP para gerar a tela.

fig. 2.5 - Programa DISSCR/ENASCR.

```
10 SCREEN2
20 DEFUSR=&H41
30 DEFUSR1=&H44
40 A=USR(1)
50 FORL=1TO100STEP3:CIRCLE(120,95),L,,,
,L/50:NEXT
60 A=USR1(1)
70 GOTO70
Ok
■
```

Este programa desabilita o VDP através da rotina DISSCR, monta uma figura na tela de alta resolução gráfica e habilita o VDP através da rotina ENASCR, mostrando a figura já pronta.

CHGCLR A rotina CHGCLR seleciona a cor dos caracteres, a cor do fundo e a cor da borda. A seleção das cores é feita por 3 variáveis do sistema. FORCLR(&HF3E9) para a cor dos caracteres, BAKCLR (&HF3EA)para a cor do fundo e BDRCLR(&HF3EB) para a cor da borda. Os valores para as variáveis do sistema devem ser atribuídos através do comando POKE e em seguida deve ser chamada a rotina.

fig. 2.6 - Programa CHGCLR.

```
10 DEFUSR=&H62
20 INPUT"COR DO FUNDO";A
30 INPUT"COR DOS CARACTERES";B
40 INPUT"COR DA BORDA";C
50 POKE&HF3EA,A
60 POKE&HF3E9,B
70 POKE&HF3EB,C
80 A=USR(1)
90 GOTO20
Ok
```

POSIT Esta rotina é semelhante ao LOCATE do BASIC e posiciona o cursor na coluna especificada pelo registrador L e na linha especificada pelo registrador H . O ponto de entrada da rotina é o endereço &H00C6.

fig. 2.7 - Programa POSIT.

```
100 CLS
110 CLEAR200,&HE0A0
120 DEFUSR=&HE0A0
130 FORL=0TO6
140 READA
150 POKE&HE0A0+L,A
160 NEXT
170 INPUT"COLUNA";B
180 INPUT"LINHA";C
190 POKE&HE0A1,B
200 POKE&HE0A2,C
210 A=USR(1)
220 PRINT"EXPERT"
230 GOTO170
240 DATA&H21,00,00,&HCD,&HC6,00,&HC9
Ok
```

Como esta rotina precisa dos dados contidos nos registradores L e H do Z80, o programa reserva espaço a partir do endereço &HE0A0 para colocar uma rotina que atribui aos registradores o valor da coluna e da linha desejadas. O programa insere o valor da coluna e da linha nos endereços &HE0A1 e &HE0A2, através do comando POKE e executa o programa em linguagem de máquina que chama a rotina POSIT.

GRPRT A rotina GRPRT imprime um caracter na tela gráfica de alta resolução (SCREEN 2) ou na tela gráfica multicor (SCREEN 3). Seu ponto de entrada é o endereço &H008D, mas ela precisa que o código do caracter esteja no registrador A. A seleção da posição de impressão do caractere nas telas gráficas é feita pelo comando PSET ou PRESET . Ele determina a posição do vértice esquerdo superior do caracter que será impresso.

fig. 2.8 - Posicionamento do caractere S com PSET ou PRESET.

Este programa apresenta, na SCREEN 2, todos os caracteres que podem ser impressos por esta rotina.

fig. 2.9 - PROGRAMA GRPRT (SCREEN 2).

```
10 CLEAR200,&HE0A0
20 DEFUSR=&HE0A0
30 SCREEN2
40 B=10
50 FOR L=0 TO 5:READ A:POKE &HE0A0+L,A:
NEXT L
60 FOR R=1 TO 255
70 POKE &HE0A1,R
80 C=C+8
90 IF R\(B/10))31 THEN B=B+10
100 IF C>248 THEN C=0
110 PRESET (C,B)
120 A=USR(1)
130 NEXT R
140 GOTO 140
150 DATA &H3E,00,&HCD,&H8D,00,&HC9
Ok
```

O programa a seguir apresenta a rotina GRPRT sendo utilizada na SCREEN 3 e funciona de forma semelhante ao anterior.

fig. 2.10 - Programa GRPRT (SCREEN 3).

```
10 CLEAR200,&HE0A0
20 DEFUSR=&HE0A0
30 SCREEN3
50 FOR L=0 TO 5:READ A:POKE &HE0A0+L,A:
NEXT L
60 FOR R=50 TO 170 STEP 30
70 READ A
80 POKE &HE0A1,A
110 PRESET (R,50)
120 A=USR(1)
130 NEXT R
140 GOTO 140
150 DATA &H3E,00,&HCD,&H8D,00,&HC9
160 DATA 65,76,69,80,72
Ok
```

FILVRM O ponto de entrada desta rotina é o endereço &H0056. Ela preenche uma determinada região da VRAM com um determinado byte. O endereço inicial de preenchimento na VRAM deverá estar no par HL, o comprimento da área a ser preenchida deverá estar no par BC, e o código do caractere deverá estar no registrador A. Este programa trabalha na SCREEN 0 e preenche a área da VRAM correspondente à tabela que contém os códigos do que está na tela, com o caractere da última tecla pressionada.

fig. 2.11 - Programa FILVRM.

```
10 CLEAR200,&HE0A0
20 FOR R=0 TO11: READ A: POKE &HE0A0+R,
A: NEXT
30 DEFUSR=&HE0A0
40 A$=INPUT$(1)
50 POKE &HE0A1,ASC(A$)
60 A=USR(1)
70 GOTO 40
80 DATA&H3E,00,&H21,00,00,01,192,3,&HCD
,&H56,00,&HC9
Ok
■
```

LDIRMV A rotina LDIRMV transfere um bloco de bytes da VRAM para a RAM e o ponto de entrada é o endereço &H0056. O endereço inicial da tabela na VRAM deve estar em HL , o endereço de destino na RAM deve estar no par DE e o comprimento da tabela deverá estar no par BC.

LDIRVM O funcionamento desta rotina é semelhante ao da LDIRMV, só que a transferencia é da RAM para a VRAM, e o ponto de entrada é o endereço &H0059. O endereço inicial da tabela na RAM deve estar no par HL, o endereço de destino da tabela na VRAM deve estar em DE e o comprimento no par BC. O programa a seguir mostra uma das formas de armazenar telas de texto na SCREEN 0. Ele cria 2 buffers de 960 bytes cada um, a partir do endereço &HA000, usando para o armazenamento a rotina LDIRMV. Como, para se utilizar essa rotina, os registradores precisam de dados, existe uma rotina definida a partir do endereço &HA000 que carrega os registradores com os devidos valores. O quinto e o sexto byte da rotina definem o endereço em que ela será armazenada. A rotina que coloca a tela novamente na VRAM começa no endereço &HA00D e utiliza a rotina LDIRVM contida no BIOS. O terceiro e o quarto byte da rotina Assembly contêm o endereço do buffer no qual está armazenada a tela. O funcionamento do programa é simples: basta pressionar F1 para obter a tela do primeiro buffer e F2 para obter a tela do segundo buffer.

fig. 2.12 - Programa LDIRMVM.

```
10 WIDTH32
20 CLS
30 CLEAR200,&HA000
40 DEFUSR=&HA000
50 DEFUSR1=&HA00D
60 '-----INSERE ROTINA LM------------
```

```
70 FOR R=0 TO 25
80 READ A
90 POKE (&HA000+R),A
100 NEXT
110 '-----MONTA TELA 1----------------
120 FORR=0TO90
130 PRINT" TELA 1 ";
140 NEXT
150 GOSUB440
160 '-----ARMAZENA TELA 1-------------
170 A=USR(0)
180 '-----MONTA TELA 2----------------
190 CLS
200 FORR=0TO90
210 PRINT" TELA 2 ";
220 NEXT
230 GOSUB440
240 '-----ARMAZENA TELA 2-------------
250 POKE&HA004,&HC0
260 POKE&HA005,&HB3
270 A=USR(0)
280 '-----CONTROLA F1 E F2------------
290 ON KEY GOSUB 340,390
```

```
300 KEY(1) ON
310 KEY(2) ON
320 GOTO320
330 '---COLOCA TELA 1 NA VRAM---------
340 POKE&HA00E,0
350 POKE&HA00F,&HB0
360 A=USR1(0)
370 RETURN
380 '---COLOCA TELA 2 NA VRAM---------
390 POKE&HA00E,&HC0
400 POKE&HA00F,&HB3
410 A=USR1(0)
420 RETURN
430 '----PERDA DE TEMPO---------------
440 FORL=0TO200
450 NEXT
460 RETURN
470 '------DADOS ROTINA LM------------
480 DATA &H21,00,00,&H11,00,&HB0,01,192
,3,&HCD,&H59,00,&HC9
490 DATA &H21,00,&HB0,&H11,00,00,01,192
,3,&HCD,&H5C,00,&HC9
Ok
■
```

WRPSG A rotina WRPSG insere um dado em um registrador do PSG[1]. O dado deve estar no registrador E do Z-80 e o número do registrador do Z-80 deve estar no registrador A do Z-80. O ponto de entrada da rotina é o endereço &H0093. O programa a seguir apresenta uma música, contida nas linhas DATA, totalmente executada através dessa rotina.

fig. 2.13 - Programa WRPSG.

```
100 '------MONTA TELA-----------------
110 CLS
120 PRINT"CANAL A   REG 0:        REG 1:
 0",,,"CANAL B   REG 2:         REG 3:",,
,"CANAL C   REG 4:         REG 5:"
130 '----MONTA ROTINA LM--------------
140 DEFUSR=&HA000
150 CLEAR200,&HA000
160 RESTORE820
170 FOR R=0 TO8
180 READ X
190 POKE(&HA000+R),X
200 NEXT
210 '----INICIALIZA CANAIS------------
220 FOR L=0 TO 7
230 READ C,D
240 POKE&HA001,C :POKE&HA003,D
250 B=USR(0)
260 NEXT
270 '-------MUSICA--------------------
280 RESTORE
290 READ F
300 IF F=256 THEN GOTO690
310 READT
320 '-------CANAL A-------------------
330 POKE&HA001,0
340 POKE&HA003,F
350 B=USR(0)
360 LOCATE 16,0
370 PRINTUSING"###";F
380 '-------CANAL B-------------------
390 FB=8*F
400 F3=FB\256
410 F2=FBMOD256
420 POKE&HA003,F2
430 POKE&HA001,2
440 B=USR(0)
450 POKE&HA003,F3
460 POKE&HA001,3
470 B=USR(0)
480 LOCATE 16,2
490 PRINTUSING"###";F2
```

```
500 LOCATE 29,2
510 PRINTUSING"##";F3
520 '-------CANAL C------------------
530 FC=4*F
540 F5=FC\256
550 F4=FCMOD256
560 POKE&HA003,F4
570 POKE&HA001,4
580 B=USR(0)
590 POKE&HA003,F5
600 POKE&HA001,5
610 B=USR(0)
620 LOCATE 16,4
630 PRINTUSING"###";F4
640 LOCATE 29,4
650 PRINTUSING"##";F5
660 FORL=0TOT*4:NEXT
670 GOTO290
680 '----------FIM-------------------
690 BEEP
700 END
710 '------DADOS MUSICA---------------
720 DATA100,60,0,1,100,20,112,20
```

```
730 DATA120,20,133,20,150,20,112,60,100
,10,90,10,100,60,0,30
740 DATA 100,60,0,1,100,20,112,20
750 DATA 120,20,133,20,150,20,84,60,75,
10,65,10,75,60,0,30
760 DATA 112,15,60,60,66,20,55,20,60,20
,66,20,75,20,66,40,90,40,135,60,0,10
770 DATA112,15,60,60,66,20,50,20,55,20,
60,20,66,20,45,120
780 DATA0,10,45,60,0,1,45,20,50,20
790 DATA 54,20,60,20,66,20,50,60,45,10,
40,10,45,60,0,10,45,60,0,1,45,20,50,20
800 DATA54,20,60,20,66,20,75,60,67,10,6
0,10,67,120,256
810 '------DADOS ROTINA LM-----------
820 DATA &H3E,00,&H11,00,00,&HCD,&H93,0
0,&HC9
830 '----DADOS INCIALIZA PSG----------
840 DATA0,0,7,56,5,0,10,8,3,0,9,8,1,0,8
,8
Ok
■
```

CARACTERES

Os caracteres disponíveis inicialmente no MSX encontram-se numa tabela da ROM entre os endereços &H1BBF e &H23BF, ocupando 2048 bytes.

Essa tabela é transferida para a VRAM sempre que as telas de texto estão sendo usadas (SCREEN 0 ou SCREEN 1). Na VRAM ela é chamada de tabela do gerador de padrões e seu endereço inicial pode ser obtido com o comando BASE(2), para a SCREEN 0, e BASE(7), para SCREEN 1. Uma vez que a tabela tenha sido transferida para a VRAM, os caracteres se tornam redefiníveis, como será explicado no capítulo 4, sobre o VDP.

Um caracter é composto por pequenos pontos acesos ou apagados chamados pixels. Em sua estrutura de oito linhas por oito colunas o caractere apresenta 64 pixels (figura 2.14).

fig 2.14 - Matriz de um caractere.

Transformando cada pixel de uma dada linha no número 1, (se ele estiver aceso) ou no número 0, (se ele estiver apagado) converteremos a linha toda num número binário entre 0 e 255 ou seja, num byte. Isso permite que o computador tenha armazenado na ROM os bytes que formam a tabela de caracteres.

O armazenamento dos bytes na tabela é feito em grupos de oito. Cada byte do octeto representa uma linha do caractere, começando pela superior.

Na tabela, a sequência dos grupos define os caracteres.

A figura 2.15 mostra os bytes correspondentes ao segundo caractere da tabela, o endereço em que ele está e o conteúdo de cada byte em decimal e em binário.

fig. 2.15 - Bytes x Caracteres.

Endereço	Decimal	Binário
&H1BC7	60	00111100
&H1BC8	66	01000010
&H1BC9	165	10100101
&H1BCA	129	10000001
&H1BCB	165	10100101
&H1BCC	153	10011001
&H1BCD	66	01000010
&H1BCE	60	00111100

O programa da figura 2.16 apresenta a forma de armazenamento de qualquer caractere, o endereço de todos os bytes que o compõe e também o conteúdo de cada byte na forma decimal e binária.

fig. 2.16 - Armazenamento dos caracteres na ROM.

```
100 CLEAR 4000
110 DEF FN B=A*8+&H1BBF+L :
120 CLS :WIDTH35 :COLOR 15,4
130 GOSUB410
140 ON ERROR GOTO 410
150 '-----ENTRA CARACTER-------------
160 LOCATE0,0
170 INPUT"CARACTER";A$
180 IFLEN(A$)=2 OR LEFT$(A$,1)=CHR$(1)
THEN A=ASC(MID$(A$,2))-64:GOTO210
190 A=ASC(A$)
200 '-----CALCULA NUMEROS------------
210 LOCATE2,4
220 FORL=0TO7
230 LOCATE2
240 B=PEEK(FN B)
250 H$=HEX$(FN B)
260 B$=STR$(B)
270 X$=BIN$(B)
280 IFLEN(X$)<8THENX$="0"+X$:GOTO280
290 '------IMPRIME NUMEROS-----------
300 PRINT H$;:LOCATE 10:PRINT USING"###
";B;: LOCATE 15 :PRINT X$;
310 LOCATE25
320 '------IMPRIME CARACTER----------
330 FORR=1TO8
340 IFMID$(X$,R,1)="1" THEN PRINTCHR$(2
19);ELSE PRINT"G";
350 NEXT R
360 PRINT
370 NEXT L
380 LOCATE0,0 :GOTO 170
390 END
400 '------IMPRIME MOLDURA-----------
410 LOCATE0,1
420 PRINTSTRING$(34,220);" ";
430 PRINTCHR$(219);"ENDERE~O";CHR$(219)
;"BYTE";CHR$(219);CHR$(219);"BINARIO";C
HR$(219);"CARACTER";CHR$(219);CHR$(219)
440 PRINTSTRING$(34,219)
450 FOR R=0 TO 7
460 PRINTCHR$(219);SPC(6);CHR$(219);SPC
(5);CHR$(219);SPC(10);CHR$(219);SPC(8);
CHR$(219)
470 NEXT R
480 PRINTSTRING$(34,223)
490 GOTO140
```

Os dois programas a seguir têm funcionamento muito semelhante. O primeiro é um ampliador de caracteres e o segundo transforma qualquer caractere num sprite, permitindo sua movimentação por toda a tela.

fig. 2.17 - Caracteres Gigantes.

```
100 SCREEN0: KEYOFF: WIDTH40
110 CLEAR200: PRINT"CARACTER ?":LOCATE2
,2:LINEINPUTA$:CLS
120 IF LEN(A$)=2 OR  LEFT$(A$,1)=CHR$(1
) THEN 130 ELSE 150
130 A=ASC(MID$(A$,2))-64
140 GOTO160
150 A=ASC(A$)
160 DEF FN B=PEEK(A*8+&H1BBF+N)
170 FORN=0TO7
180 LETX$=BIN$(FN B) :IFLEN(X$)=8THENGO
TO200
190 X$="0"+X$:IFLEN(X$)=8THENGOTO200ELS
EGOTO190
200  FORL=1TO8
210  IFMID$(X$,L,1 )="1"THENGOSUB250
220  NEXTL
230 NEXTN
240 LOCATE0,0: GOTO110
250 LOCATE(L*3)+8,(N*3):PRINTSTRING$(3,
219)
260 LOCATE(L*3)+8,((N*3)+1):PRINTSTRING
$(3,219)
270 IFN<7THEN LOCATE(L*3)+8,((N*3)+2):P
RINTSTRING$(3,219)
280 RETURN
```

fig. 2.18 - Programa para transformar caracteres em sprites.

```
10 SCREEN 1,1
20 PRINT"AS SETAS CONTROLAM O SPRITE."
30 PRINT"A BARRA DE ESPACO SELECIONA  O
UTRO CARACTERE."
40 '-------------ENTRA CARACTERE-------
50 CLEAR 200: LOCATE 0,3: INPUT"CARACTE
RE ";A$
60 '---------CALCULA CODIGO------------
70 IF LEN(A$)=2 OR LEFT$(A$,1)=CHR$(1)T
HEN A=ASC(MID$(A$,2))-64 :GOTO 90
80 A=ASC(A$)
90 DEF FN B=PEEK(A*8+&H1BBF+N)
100 '---------MONTA SPRITE--------------
110 FOR N=0 TO 7
120 X$=X$+CHR$(FN B)
130 NEXT N
140 SPRITE$(1)=X$
150 '---------CONTROLA SPRITE-----------
```

```
160 C=128
170 L=96
180 PUTSPRI)E1,(C,L),4,1
190 Z=ASC(INPUT$(1))
200 IF Z=28 THENC=C+1
210 IF Z=29 THENC=C-1
220 IF Z=30 THENL=L-1
230 IF Z=31 THENL=L+1
240 IF Z=32 THEN 50
250 GOTO 180
Ok
■
```

Diferenças na tabela de caracteres do Expert e do Hot-Bit

Como o sistema MSX é um padrão mundial ao qual aderiram fabricantes de diferentes continentes, parte da tabela de caracteres é arbitrária. Nós usamos como exemplo as tabelas de caracteres dos MSX Expert e Hot-Bit. O Hot-Bit tem a tabela ligeiramente diferente da do Expert a partir do caractere de código 126. As diferenças são mais notórias nos caracteres acentuados. Em certos caracteres apenas a forma da letra é diferente, o que não representa nenhum problema para o usuário. Existem, entretanto, alguns caracteres de mesmo código que, nos dois micros representam letras ou símbolos completamente diferentes.

Isso dificulta um pouco a compreensão das mensagens de um programa elaborado num micro e executado no outro. A figura 2.19 mostra algumas diferenças entre os caracteres do Hot-Bit e do Expert 1.0.

fig. 2.19 - Algumas diferenças de caracteres entre o HOTBIT e o Expert versão 1.0.

COD	Exp.	H.B.
96	Ç	`
126	X	~
128		Ç
129	ü	ü
132	ä	Á
134	à	
135		ç
137	ë	í
138	è	Ó
139	ï	Ú
140	î	Â
141	ì	Ê
142	Ä	Ô
143	Å	À
147	ô	ô
148	ö	ö
149	ò	ó
151	ù	ú
152	ÿ	ý
153	Ö	Ö
154	Ü	Ü
158	Cr	Pt
162	ó	ó
163	ú	ú
166	ª	ª
167	º	º
174	«	«
175	»	»
181	õ	õ
182	Õ	Õ
185	ÿ	ÿ
189	%	%
240	≅	≡

Obs.: Atualmente, os dois computadores estão totalmente compatibilizados em seus conjuntos de caracteres conforme o padrão ABNT (veja as páginas 10 e 11).

Um software elaborado num desses dois micros sempre apresentará essas falhas se imprimir os caracteres diferentes e a única maneira de mudar isso é a alteração de todas as mensagens do programa. Uma dos meios é a edição de linha por linha do programa, o que é muito trabalhoso. Outra maneira é utilizar a seguinte rotina.

fig. 2.20 - Programa adaptador.

```
65000 DIM A(6,1)
65010 FOR N=0 TO 5 : READ H,E
65020 A(N,0)=H:A(N,1)=E:NEXT N
65030 FOR R=32767 TO 65535!
65040 K=PEEK(R)
65050 IF K=0 AND PEEK(R+1)=0 AND PEEK(R
+2)=0 THEN PRINT"PROGRAMA TRANSFORMADO"
: END
65060 IF K=0 THEN R=R+4 :F=0 :GOTO 6510
0 ELSE GOTO65070
65070 IF K=143 THEN F=-1
65080 IF K=34 THEN F=NOTF
65090 IF F=-1 THEN GOTO65120
65100 NEXT R : END
65120 FORM=0 TO 5
65130 IF PEEK(R)=A(M,0) THEN POKE R,A(M
,1):PRINTR;".....";PEEK(R) :GOTO65100
65140 NEXT M : GOTO 65100
65150 DATA 135,96,128,126,126,128,183,1
29,96,135,132,65
```

Esta rotina adapta programas em BASIC feitos para o Hot-Bit, deixando seus caracteres compatíveis com o EXPERT. Ela deve ser sempre gravada no formato ASCII (veja capítulo 6), pois assim, poderá ser incluida no final do programa a ser adaptado, através do comando MERGE.

Fica como exercício para o leitor, elaborar um programa que faça o inverso.

Palavras Reservadas

Para entender BASIC, o MSX dispõe de 163 palavras armazenadas na ROM entre os endereços &H3A72 e &H3D22. Além dessas palavras (reservadas ao BASIC), existem 10 caracteres (também reservados ao BASIC) entre os endereços &H3D26 e &H3D39.

Essas 163 palavras e os 10 caracteres são necessários para que, ao ler um programa em BASIC na RAM, o interpretador possa identificá-

los. Cada palavra e cada caractere reservados tem associados a si um código ou token entre 0 e 255. No capítulo 1 você já viu para que servem esses códigos (lembra-se da piada dos loucos?). Agora vamos ver como eles estão armazenados junto às palavras e caracteres reservados.

A tabela das palavras está armazenada de forma compactada, iniciando com as que começam por A (AUTO, AND , etc.) e indo, em ordem alfabética, até a palavra XOR. A ordem alfabética não é usada para ordenar palavras que começam pela mesma letra!

O programa listado na figura 2.21 gera uma tabela com todas as palavras reservadas, seus respectivos códigos (tokens), seus endereços iniciais determinando, ainda, se se trata de comandos ou de funções.

fig. 2.21 - Programa para listar as palavras reservadas.

```
100 ' Palavras Reservadas
110 EN=14962
120 I=65
130 PRINT EN;CHR$(I);
140 P=PEEK(EN)
150 Q=PEEK(EN+1)
160 P$=CHR$(P)
170 IF P<128 THEN PRINT P$;:GOTO 220
180 PRINTCHR$(P-128);TAB(14);Q;TAB(20);
190 IF Q<128 THEN PRINT "funcao" ELSE P
RINT "comando"
200 EN=EN+1
210 IF PEEK(EN+1)<>0 THEN PRINT EN+1;CH
R$(I);
220 IF PEEK(EN)<>0 THEN 260
230 I=I+1:Q$=CHR$(I)
240 IF Q$="J" OR Q$="Q" THEN 260
250 PRINT EN+1;Q$;
260 EN=EN+1
270 IF EN<=15649 THEN 140
```

Ao executá-lo, você obterá no vídeo a tabela da figura 2.22 .

fig. 2.22 - Palavras Reservadas.

END.	PALAVRA	TOKEN	TIPO
14962	AUTO	169	comando
14966	AND	246	comando
14969	ABS	6	funcao
14972	ATN	14	funcao
14975	ASC	21	funcao
14978	ATTR$	233	comando
14984	BASE	201	comando
14988	BSAVE	208	comando
14993	BLOAD	207	comando
14998	BEEP	192	comando
15002	BIN$	29	funcao
15007	CALL	202	comando
15011	CLOSE	180	comando
15016	COPY	214	comando
15020	CONT	153	comando
15024	CLEAR	146	comando
15029	CLOAD	155	comando
15034	CSAVE	154	comando

15039	CSRLIN	232	comando
15045	CINT	30	funcao
15049	CSNG	31	funcao
15053	CDBL	32	funcao
15057	CVI	40	funcao
15060	CVS	41	funcao
15063	CVD	42	funcao
15066	COS	12	funcao
15069	CHR$	22	funcao
15073	CIRCLE	188	comando
15079	COLOR	189	comando
15084	CLS	159	comando
15087	CMD	215	comando
15091	DELETE	168	comando
15097	DATA	132	comando
15101	DIM	134	comando
15104	DEFSTR	171	comando
15110	DEFINT	172	comando
15116	DEFSNG	173	comando
15122	DEFDBL	174	comando
15128	DSKO$	209	comando
15133	DEF	151	comando
15136	DSKI$	234	comando
15141	DSKF	38	funcao
15145	DRAW	190	comando
15150	ELSE	161	comando
15154	END	129	comando
15157	ERASE	165	comando
15162	ERROR	166	comando
15167	ERL	225	comando
15170	ERR	226	comando
15173	EXP	11	funcao
15176	EOF	43	funcao
15179	EQV	249	comando
15183	FOR	130	comando
15186	FIELD	177	comando
15191	FILES	183	comando
15196	FN	222	comando
15198	FRE	15	funcao
15201	FIX	33	funcao
15204	FPOS	39	funcao
15209	GOTO	137	comando
15213	GO TO	137	comando
15218	GOSUB	141	comando
15223	GET	178	comando
15227	HEX$	27	funcao
15232	INPUT	133	comando
15237	IF	139	comando
15239	INSTR	229	comando
15244	INT	5	funcao
15247	INP	16	funcao
15250	IMP	250	comando
15253	INKEY$	236	comando
15259	IPL	213	comando
15264	KILL	212	comando
15268	KEY	204	comando
15272	LPRINT	157	comando
15278	LLIST	158	comando
15283	LPOS	28	funcao
15287	LET	136	comando
15290	LOCATE	216	comando
15296	LINE	175	comando
15300	LOAD	181	comando
15304	LSET	184	comando
15308	LIST	147	comando
15312	LFILES	187	comando
15318	LOG	10	funcao
15321	LOC	44	funcao
15324	LEN	18	funcao
15327	LEFT$	1	funcao
15332	LOF	45	funcao
15336	MOTOR	206	comando
15341	MERGE	182	comando
15346	MOD	251	comando
15349	MKI$	46	funcao
15353	MKS$	47	funcao
15357	MKD$	48	funcao
15361	MID$	3	funcao
15365	MAX	205	comando
15369	NEXT	131	comando
15373	NAME	211	comando
15377	NEW	148	comando
15380	NOT	224	comando
15384	OPEN	176	comando
15388	OUT	156	comando
15391	ON	149	comando
15393	OR	247	comando
15395	OCT$	26	funcao
15399	OFF	235	comando
15403	PRINT	145	comando
15408	PUT	179	comando
15411	POKE	152	comando
15415	POS	17	funcao
15418	PEEK	23	funcao
15422	PSET	194	comando
15426	PRESET	195	comando
15432	POINT	237	comando
15437	PAINT	191	comando
15442	PDL	36	funcao
15445	PAD	37	funcao
15448	PLAY	193	comando
15454	RETURN	142	comando
15460	READ	135	comando
15464	RUN	138	comando
15467	RESTORE	140	comando
15474	REM	143	comando

15477	RESUME	167	comando
15483	RSET	185	comando
15487	RIGHT$	2	funcao
15493	RND	8	funcao
15496	RENUM	170	comando
15502	SCREEN	197	comando
15508	SPRITE	199	comando
15514	STOP	144	comando
15518	SWAP	164	comando
15522	SET	210	comando
15525	SAVE	186	comando
15529	SPC(	223	comando
15533	STEP	220	comando
15537	SGN	4	funcao
15540	SQR	7	funcao
15543	SIN	9	funcao
15546	STR$	19	funcao
15550	STRING$	227	comando
15557	SPACE$	25	funcao
15563	SOUND	196	comando
15568	STICK	34	funcao
15573	STRIG	35	funcao
15579	THEN	218	comando
15583	TRON	162	comando
15587	TROFF	163	comando
15592	TAB(	219	comando
15596	TO	217	comando
15598	TIME	203	comando
15602	TAN	13	funcao
15606	USING	228	comando
15611	USR	221	comando
15615	VAL	20	funcao
15618	VARPTR	231	comando
15624	VDP	200	comando
15627	VPOKE	198	comando
15632	VPEEK	24	funcao
15638	WIDTH	160	comando
15643	WAIT	150	comando
15648	XOR	248	comando

Para economizar memória, a primeira letra de cada palavra não é armazenada. Veja, por exemplo, como as palavras iniciadas por A e B estão na memória (fig. 2.23).

fig. 2.23 - Exemplo de palavras iniciadas por A e B.

```
                     A
&H3A72    &H55       U
&H3A73    &H54       T
&H3A74    &HCF       O = CHR$(&HCF-&H80)
&H3A75    &HA9         = token (COMANDO)
-----------------------------------------------
                     A
&H3A76    &H4E       N
&H3A77    &HC4       D = CHR$(&HC4-&H80)
&H3A78    &HF6         = token (COMANDO)
-----------------------------------------------
                     A
&H3A79    &H42       B
&H3A7A    &HD3       S = CHR$(&HD3-&H80)
&H3A7B    &H06         = token (FUNCAO)
-----------------------------------------------
                     A
&H3A7C    &H54       T
&H3A7D    &HCE       N = CHR$(&HCE-&H80)
&H3A7E    &H0E         = token (FUNCAO)
-----------------------------------------------
                     A
&H3A7F    &H53       S
&H3A80    &HC3       C = CHR$(&HC3-&H80)
&H3A81    &H15         = token (FUNCAO)
-----------------------------------------------
```

```
                    A
&H3A82    &H54      T
&H3A83    &H54      T
&H3A84    &H52      R
&H3A85    &HA4      $ = CHR$(&HA4-&H80)
&H3A86    &HE9        = token (COMANDO)
-----------------------------------------------
&H3A87    &H00        muda letra inicial
-----------------------------------------------
                    B
&H3A88    &H41      A
&H3A89    &H53      S
&H3A8A    &HC5      E = CHR$(&HC5-&H80)
&H3A8B    &HC9        = token (COMANDO)
-----------------------------------------------
                    B
&H3A8C    &H53      S
&H3A8D    &H41      A
&H3A8E    &H56      V
&H3A8F    &HC5      E = CHR$(&HC5-&H80)
&H3A90    &HD0        = token (COMANDO)
-----------------------------------------------
                    B
&H3A91    &H4C      L
&H3A92    &H4F      O
&H3A93    &H41      A
&H3A94    &HC4      D = CHR$(&HC4-&H80)
&H3A95    &HCF        = token (COMANDO)
-----------------------------------------------
                    B
&H3A96    &H45      E
&H3A97    &H45      E
&H3A98    &HD0      P = CHR$(&HD0-&H80)
&H3A99    &HC0        = token (COMANDO)
-----------------------------------------------
                    B
&H3A9A    &H49      I
&H3A9B    &H4E      N
&H3A9C    &HA4      $ = CHR$(&HA4-&H80)
&H3A9D    &H1D        = token (FUNCAO)
-----------------------------------------------
&H3A9E    &H00        muda letra inicial
-----------------------------------------------
```

Note que o último byte de cada palavra contém &H80 mais o código de seu último caractere. Na palavra AUTO, por exemplo, o último byte contém &H4F (código ASCII da letra O) mais &H80 (128 em decimal, indicando "fim de palavra").

Separando um grupo de palavras iniciado por uma letra, de outro grupo iniciado por outra letra, existe um byte com &H00.

No fim dessa tabela, indicando seu término, existem dois bytes consecutivos com &H00.

Em geral, para saber se uma palavra é uma função ou um comando basta verificar seu código. Se ele for menor que &H80 (128 em decimal), provavelmente ela é uma função; caso contrário, ela deve ser um comando.

A tabela com os 10 caracteres reservados está organizada como mostra a figura 2.24 , juntamente com um programa que a produz no vídeo ao ser executado.

fig. 2.24 - Caracteres reservados.

```
10 PRINT "ENDERECO   CARACTERE   TOKEN
"
20 FOR F=&H3D26 TO &H3D39 STEP 2
30 G=PEEK(F):H=PEEK(1+F)
40 PRINT " &H";HEX$(F);"          ";
50 IF G>127 THEN PRINT CHR$(G-128);
60 IF G<128 THEN PRINT CHR$(G);
70 PRINT"          &H";HEX$(H)
80 NEXT F
```

```
run
ENDERECO   CARACTERE   TOKEN

 &H3D26        +         &HF1
 &H3D28        -         &HF2
 &H3D2A        *         &HF3
 &H3D2C        /         &HF4
 &H3D2E        ^         &HF5
 &H3D30        \         &HFC
 &H3D32        '         &HE6
 &H3D34        >         &HEE
 &H3D36        =         &HEF
 &H3D38        <         &HF0
Ok
■
```

Mensagens de Erros

O MSX dispõe de 38 mensagens que auxiliam a descoberta de erros nos programas. Essas mensagens estão armazenadas em uma tabela situada entre os endereços &H3D75 e &H3FE1 ocupando 621 bytes.

Estão contidas na tabela todas as mensagens de erro e elas são apresentadas apenas nas telas de texto (SCREEN 0 e SCREEN 1).

Vejamos como as mensagens estão na ROM (fig. 2.25).

fig. 2.25 - Programa para listar as mensagens de erro.

```
10 ' Com este programa voce pode
20 ' observar como as mensagens de
30 ' erro estao armazenadas na ROM !
40 FOR R=&H3D75 TO &H3FE1
50 IFPEEK(R)=0THENPRINTSTRING$(34,45)
60 PRINT"&H";HEX$(R);" ... ";:PRINTUSIN
G"###";PEEK(R);:PRINT" ... ";CHR$(PEEK(
R))
70 IFPEEK(R)=0THENPRINTSTRING$(34,45)
80 NEXT R
```

```
                  ----------------------------------
início  ───►      &H3D75 ...   0 ...
                  ----------------------------------
                  &H3D76 ...  78 ... N
                  &H3D77 ...  69 ... E
                  &H3D78 ...  88 ... X
                  &H3D79 ...  84 ... T
                  &H3D7A ...  32 ...
                  &H3D7B ... 119 ... w
                  &H3D7C ... 105 ... i
                  &H3D7D ... 116 ... t
                  &H3D7E ... 104 ... h
                  &H3D7F ... 111 ... o
                  &H3D80 ... 117 ... u
                  &H3D81 ... 116 ... t
                  &H3D82 ...  32 ...
                  &H3D83 ...  70 ... F
                  &H3D84 ...  79 ... O
                  &H3D85 ...  82 ... R
  muda            ----------------------------------
   a    ───►      &H3D86 ...   0 ...
mensagem          ----------------------------------
                  &H3D87 ...  83 ... S
                  &H3D88 ... 121 ... y
                  &H3D89 ... 110 ... n
                  &H3D8A ... 116 ... t
                  &H3D8B ...  97 ... a
                  &H3D8C ... 120 ... x
                  &H3D8D ...  32 ...
                  &H3D8E ... 101 ... e
                  &H3D8F ... 114 ... r
                  &H3D90 ... 114 ... r
                  &H3D91 ... 111 ... o
                  &H3D92 ... 114 ... r
```

```
   muda        ----------------------------------------
     a    ---> &H3D93 ...   0 ...
 mensagem      ----------------------------------------
               &H3D94 ...  82 ... R
               &H3D95 ...  69 ... E
               &H3D96 ...  84 ... T
               &H3D97 ...  85 ... U
               &H3D98 ...  82 ... R
               &H3D99 ...  78 ... N
               &H3D9A ...  32 ...
               &H3D9B ... 119 ... w
               &H3D9C ... 105 ... i
               &H3D9D ... 116 ... t
               &H3D9E ... 104 ... h
               &H3D9F ... 111 ... o
               &H3DA0 ... 117 ... u
               &H3DA1 ... 116 ... t
               &H3DA2 ...  32 ...
               &H3DA3 ...  71 ... G
               &H3DA4 ...  79 ... O
               &H3DA5 ...  83 ... S
               &H3DA6 ...  85 ... U
               &H3DA7 ...  66 ... B
   muda        ----------------------------------------
     a    ---> &H3DA8 ...   0 ...
 mensagem      ----------------------------------------
                  :           :
                 etc         etc
```

Obs: No HOTBIT essas mensagens estão em português!

O byte zerado no final de cada mensagem indica o seu término e o início da próxima.

Existem 255 códigos de erro mas apenas 38 mensagens. O código de uma mensagem é obtido pelo comando ERR e a linha que a produziu é obtida por ERL. Se a mensagem de erro foi produzida durante um comando direto, o valor de ERL é 65535.

Para as primeiras 25 mensagens o valor da função ERR é igual à ordem da mensagem na tabela. Por exemplo, quando um erro de sintaxe ocorre, aparece a seguinte mensagem:

```
Syntax error
```

Ao pedir o código do erro através da funcão ERR, o valor 2 será obtido, pois é essa a posição da mensagem na tabela.

Os códigos de erro entre 26 e 49 não possuem nenhuma mensagem na tabela e são reservados para futuras expansões do BASIC. Entretanto, quando o MSX não está conectado com uma dessas expansões, a mensagem apresentada é a do erro 23:

```
Unprintable error
```

A existência de 24 códigos sem mensagens correspondentes faz com que as 12 mensagens seguintes tenham o código de erro dado por sua ordem na tabela acrescida de 24. Por exemplo, quando o comando OPEN é executado em um arquivo já aberto, surge a seguinte mensagem:

```
File already open
```

Essa mensagem ocupa o trigésimo lugar na tabela mas seu código de erro é 54.

Os códigos de erro de 60 até 255 também não possuem mensagens correspondentes na tabela. Eles são reservados para que o próprio usuário defina as suas mensagens de erro.

Isso pode ser feito através dos comandos ERROR e ON ERROR GOTO, que devem ser combinados para acusar o erro que se deseja e para desviar o programa para uma rotina que tome alguma providência (por exemplo, que imprima uma mensagem de erro).

Caso um erro seja produzido pelo comando ERROR e o comando ON ERROR não esteja ativo para desviar a execução do programa, a mensagem de código 23 (Unprintable error) será apresentada.

As três mensagens finais da tabela não são mensagens de erro, elas completam essas mensagens ou indicam outros estados do micro.

São elas:

"in" - Inserido após a mensagem para indicar a linha em que o erro ocorreu

"Ok" - Indicando que o micro está a espera de um comando no modo direto.

"Break" - Indicando a parada de execução de um programa.

Com o programa da figura 2.26, você poderá obter no vídeo a tabela das mensagens de erros com seus respectivos códigos.

fig. 2.26 - Programa para listar mensagens.

```
10 FOR R=&H3D75 TO &H3FE1
20 IF PEEK(R)<>0 THEN 70
30 PRINT : A=A+1
40 PRINT "&H";HEX$(R+1);
50 IF A=26 THEN A=50
60 PRINTUSING"#####";A;:PRINT TAB(12);
70 PRINT CHR$(PEEK(R));
80 NEXT
```

```
&H3D76     1 NEXT without FOR
&H3D87     2 Syntax error
&H3D94     3 RETURN without GOSUB
&H3DA9     4 Out of DATA
&H3DB5     5 Illegal function call
&H3DCB     6 Overflow
&H3DD4     7 Out of memory
&H3DE2     8 Undefined line number
&H3DF8     9 Subscript out of range
&H3E0F    10 Redimensioned array
&H3E23    11 Division by zero
&H3E34    12 Illegal direct
&H3E43    13 Type mismatch
&H3E51    14 Out of string space
&H3E65    15 String too long
&H3E75    16 String formula too complex
&H3E90    17 Can't CONTINUE
&H3E9F    18 Undefined user function
&H3EB7    19 Device I/O error
&H3EC8    20 Verify error
&H3ED5    21 No RESUME
&H3EDF    22 RESUME without error
```

```
           &H3EF4    23 Unprintable error
           &H3F06    24 Missing operand
salto de → &H3F16    25 Line buffer overflow
25 para 50 &H3F28    50 FIELD overflow
           &H3F3A    51 Internal error
           &H3F49    52 Bad file number
           &H3F59    53 File not found
           &H3F68    54 File already open
           &H3F7A    55 Input past end
           &H3F89    56 Bad file name
           &H3F97    57 Direct statement in file
           &H3FB0    58 Sequential I/O only
           &H3FC4    59 File not OPEN
           &H3FD2    60  in
           &H3FD7    61 Ok

           &H3FDC    62 Break
           &H3FE2    63
```

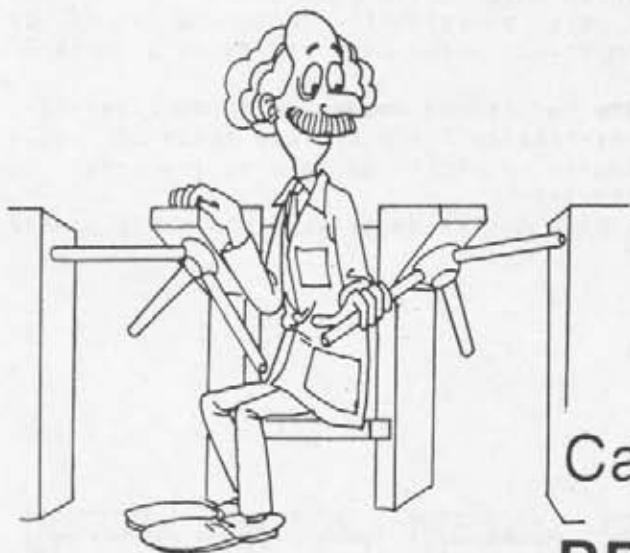
Uma das características da maioria dos micros MSX é o uso das mensagens de erro em Inglês, o que pode dificultar a perfeita compreensão do porquê de um erro.

Existem várias formas de se transformar as mensagens de erro do Inglês para o Português. Uma delas está exemplificada no programa da figura 2.27 , que utiliza as funções do BASIC (ON ERROR, ERR, ERL) para detectar e traduzir as mensagens de erro. Esta rotina pode ser inserida no final de um programa BASIC que esteja sendo digitado e testado. Para ativar a rotina basta digitar GOTO 50000 e pronto.

fig. 2.27 - Tradução das mensagens.

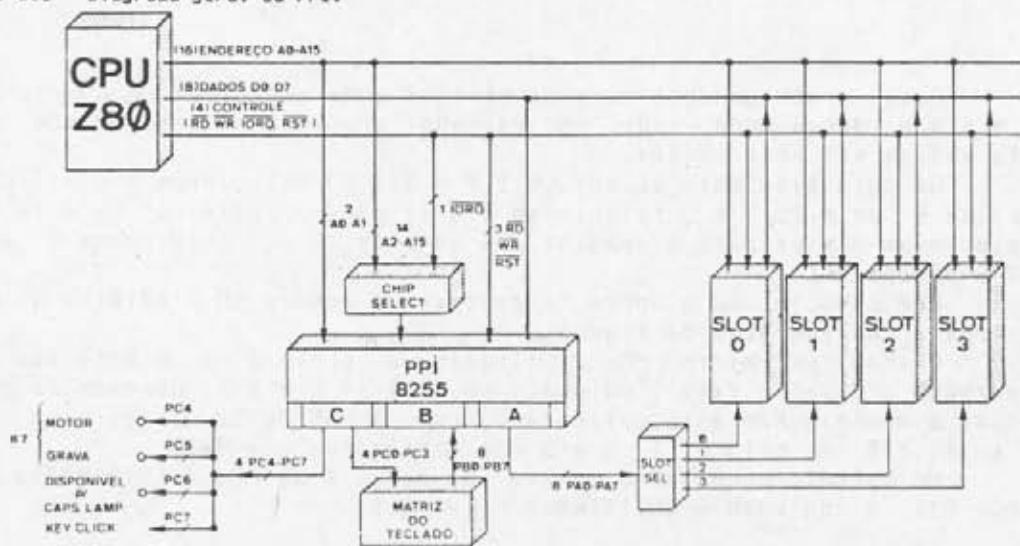
```
50000 ONERRORGOTO50010:END
50010 A=ERR
50020 IFA>25ANDA<50ORA>59THENA=23
50030 IF A>=50 THEN A=A-24
50040 RESTORE
50050 FOR R=1 TO 59
50060 READ A$
50070 IF A=R THEN GOTO50090
50080 NEXTR
50090 PRINTA$;
50100 B=ERL
50110 IF B<>65535! THEN PRINT" na";ERL
50120 END
50130 DATA NEXT sem FOR,Erro de Sintaxe
,RETURN sem GOSUB,Termino de Dados,Func
ao Ilegal,Sobrecarga,Falta Memoria,Linh
a Indefinida,Subscrito Invalido,Matriz
Redimensionada
50140 DATA Divisao por Zero,Comando Dir
eto Ilegal,Tipo de Variavel Invalido,Fa
lta Espaco Para Strings,String Muito Lo
nga,Formula de String Muito Complexa,Im
possivel,Funcao Indefinida,Erro de E/S,
Nao Gravou,Falta RESUME,RESUME Sem Erro
,Erro Nao Imprimivel
```

```
50150 DATA Excedeu Buffer,Excedeu Campo
,Erro Interno,Numero de Arquivo Invalid
o,Arquivo Nao Encontrado,Arquivo Ja Abe
rto,INPUT Apos fim,Nome de Arquivo Inva
lido,Comando Direto no Arquivo,So E/S S
equencial,Arquivo nao Aberto,Erro Nao I
mprimivel
Ok
█
```

Como exercício, o leitor pode alterar as linhas DATA colocando mensagens humorísticas ou personalizadas. Os usuários do Hot-Bit não precisam, obviamente, traduzir as mensagens para o português, mas podem também se divertir, personalizando as mensagens de erro.

Interpretador Basic

Tudo que vimos dos 32 Kbytes da ROM até aqui, estava nos primeiros 16 Kbytes (página 0). Com exceção de alguns caracteres e das mensagens de erro, o conteúdo desses primeiros 16 Kbytes permanece praticamente o mesmo em todos os micros MSX. O mesmo não se pode dizer com relação aos 16 Kbytes de &H4000 a &H7FFF, onde está armazenado o interpretador Basic e a mensagem de identificação do fabricante. Esses 16 Kbytes são mutáveis e às vezes, um mesmo fabricante pode ter no mercado duas versões distintas, com interpretadores Basic diferentes. Obviamente, o resultado final de todos os interpretadores é sempre compatível com o padrão MSX, isto é, eles interpretam o BASIC MSX! As pequenas diferenças podem estar na economia de 1 byte de memória ou de 1 centésimo de segundo na execução de um comando.

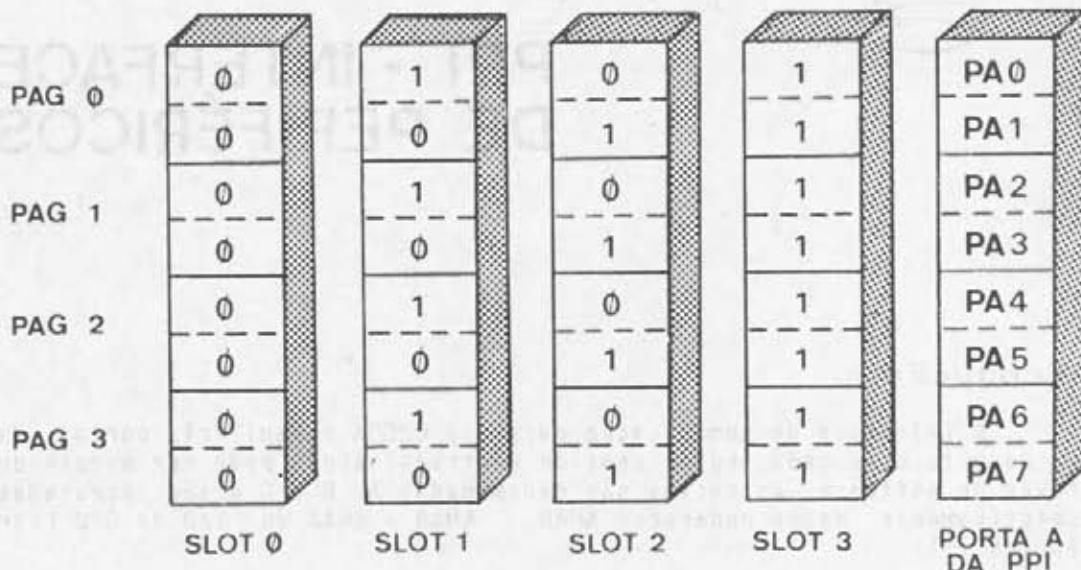
Devido a essas variações, não é conveniente usar rotinas do interpretador como sub-rotinas de programas em BASIC ou em ASSEMBLY. Nós não as comentaremos aqui, mas se mesmo assim você estiver interessado em analisá-las, no apêndice III apresentamos um programa disassemblador (em BASIC!) com o qual o interpretador pode ser pesquisado. Para usá-lo, entretanto, é conveniente que você saiba ao menos ler os mnemônicos do Z80.

Para quem quiser, mesmo com os inconvenientes, usar rotinas do interpretador, no livro PROGRAMAÇÃO AVANÇADA EM MSX os autores mostram como achar o ponto de entrada de todas elas a partir das "tokens" e comentam algumas apenas a título de ilustração.

Capítulo 3

PPI - INTERFACE DE PERIFÉRICOS

Introdução

A interface de comunicação paralela 8255A possui três portas de I/O de oito bits cada, cujo sentido (entrada/saída) pode ser escolhido através de software. As portas são denominadas A, B e C e são acessadas respectivamente pelos enderecos &HA8 &HA9 e &HAA do I/O da CPU (ver apêndice III).

fig. 3.1 - Diagrama geral da PPI.

A programação das portas A, B e C para leitura (entrada) ou para gravação (saída) é realizada através de uma quarta porta, específica para controle, acessada pelo endereço &HAB do I/O da CPU.

Quando o micro é ligado, ele inicialmente "programa" a PPI de modo que as portas A e C fiquem selecionadas como saída, e a porta B, como entrada.

As portas A, B e C podem ser reprogramadas de forma diferente, entretanto, o resto do circuito eletrônico foi projetado tendo em vista a programação inicial e, ao mudá-la, o micro deixará de funcionar, ou pior ainda, poderá até ser danificado!

A porta A é responsável pela lógica de seleção dos slots e está organizada conforme a figura 3.2 .

fig. 3.2 - Organização da Porta A.

Cada grupo de dois bits da porta A pode registrar um número entre 0 e 3 e, desse modo, pode indicar qual o número do slot em que uma certa página vai estar ativa.

Os dois bits mais baixos (bit 0 e bit 1) selecionam o slot para a página 0, os bits 2 e 3 selecionam o slot para a página 1, os bits 4 e 5 selecionam o slot para a página 2 e os bits 6 e 7 selecionam o slot para a página 3.

Por exemplo, se a porta A contiver o número &B 01001011, o micro terá a configuração da figura 3.3.

Quando um micro MSX é ligado, a porta A é programada com &Hxxxx0000 . O valor xxxx , colocado nos bits 4,5,6 e 7, depende do slot no qual a memória RAM está colocada e, como a ROM de 32 Kbytes está sempre no slot 0, os bits 0, 1, 2 e 3 são carregados com 0000.

No EXPERT, o conteúdo inicial da porta A da PPI é &B10100000 , e no HOT-BIT, o conteúdo é &B11110000 (fig. 3.4).

fig. 3.3 - Exemplo de configuração de memória.

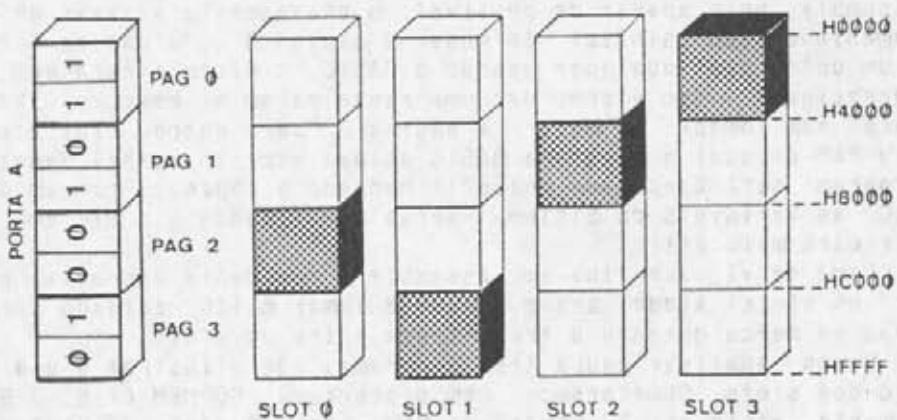
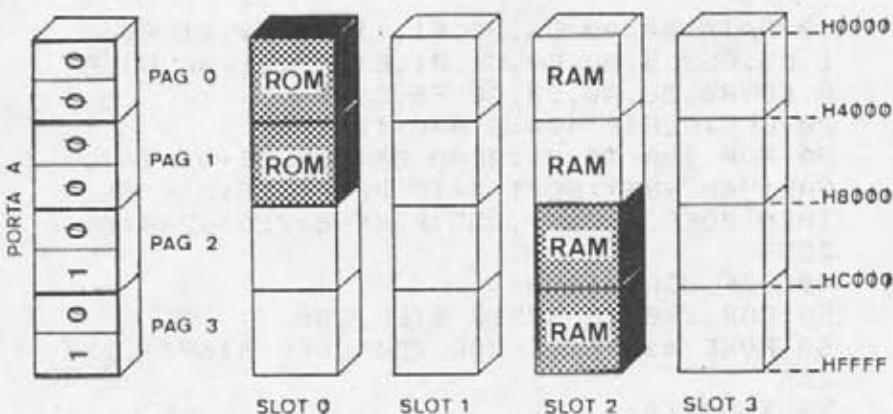
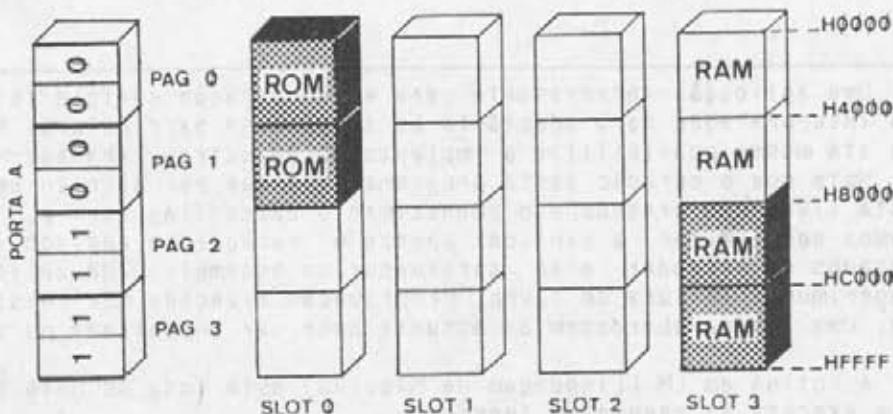



fig. 3.4 - Porta A e configuração de memória do Expert e do Hot-Bit.

EXPERT

HOTBIT

Chaveamento de slots

Para fazer o chaveamento dos slots é necessária uma sub-rotina em Assembly, pois apesar de possível, o chaveamento através do BASIC é extremamente desaconselhável. Ao mudar a página 0 ou a página 1 do slot 0 para um outro slot qualquer usando o BASIC, o micro ficará sem o Sistema Operacional. Isso é como dar uma rasteira em si mesmo e fatalmente resultará num tombo! Ao mudar a página 2 para outro slot através do BASIC, a RAM na qual o programa BASIC estava escrito estará desativada! Qual programa será executado então?!! Mudando a página 3 com um programa em BASIC, as Variáveis do Sistema serão desativadas e o Sistema Operacional ficará maluco!!!

Como se vê, a rotina em Assembly é realmente essencial para se chavear os slots! Ainda assim, deve-se tomar muito cuidado para que o micro não se perca durante o troca-troca entre os slots.

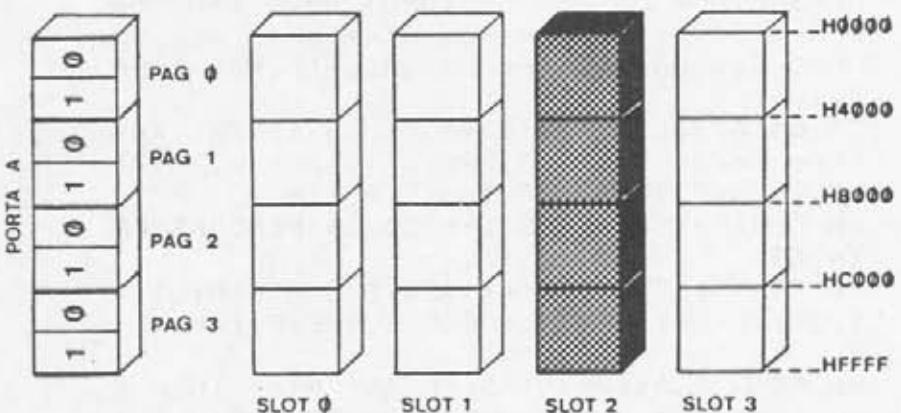
Vamos analisar agora três programas que ilustram o uso do chaveamento dos slots. Começaremos com o programa COPMEM (fig. 3.5), que simplesmente copia os 32 Kbytes de ROM (slot 0) nos 32 Kbytes de RAM inferiores do slot 2 e o deixa ativado. Como resultado, o Interpretador BASIC passa a operar em RAM e, desse modo, pode ser alterado através de POKEs.

fig. 3.5 - Programa COPMEM.

```
10 DATA 00,00,2A,00,F1,11,00,F0,01,00,0
1,D5,E5,C5,ED,B0,C1,D1,E1,F3,3E,AA,D3,A
8,ED,B0,3E,A0,D3,A8,FB,C9
20 CLS:CLEAR 400,61440!:K=170
30 FOR I=0 TO 31:READ B$:POKE I+61696!,
VAL("&H"+B$):NEXT I:IF PEEK(32513)<>71
THEN POKE 61717!,255:POKE 61723!,240:K=
255
40 DEF USR=61698!
50 FOR I=0 TO 32512 STEP 256
60 POKE 61696!,I MOD 256:POKE 61697!,I\
256
70 X=USR(0)
80 NEXT I
90 LOCATE 0,10:PRINT "RAM ATIVA EM 0000
-7FFF COM MSX BASIC":OUT 168,K
```

Uma aplicação interessante para esta operação seria alterar partes do interpretador para adaptá-lo às exigências particulares do usuário, ou até mesmo possibilitar a implantação de outras linguagens.

Note que o coração deste programa teve que ser escrito em Assembly. Este livro não pressupõe o conhecimento dessa linguagem e, portanto, vamos nos limitar a explicar apenas o raciocínio adotado. Para os interessados em entender e se aprofundar no Assembly Z80 aplicado ao MSX, sugerimos a leitura do livro "Programação Avançada MSX" desta mesma editora. Uma rápida abordagem do assunto pode ser encontrada no apêndice III.

A rotina em LM (Linguagem de Máquina) está toda no DATA da linha 10 e ela executa as seguintes funções:

a) Copia 256 bytes da ROM para uma área alta da RAM (página 3);
b) Chaveia as páginas 0 e 1 do slot 0 para o slot 2(no Expert) ou para o slot 3 (no Hot-Bit);
c) Copia os 256 bytes armazenados na página 3 da RAM para a página 0 ou 1 ;
d) Retorna as páginas 0 e 1 para o slot 0 e volta ao BASIC.

Isso tudo se faz necessário pois é impossível, num mesmo instante, ligar mais de 64 Kbytes às linhas de endereçamento do do Z80. A sub-rotina em LM copia apenas 256 bytes por vez porque as páginas 2 e 3 da RAM já estão ocupadas pelo próprio COPMEM e mais as Variáveis do Sistema. Ela deve, portanto, ser repetida até que os 32 Kbytes da ROM tenham sido transferidos.

O controle desta repetição é feito pelo BASIC, que além de servir de contador, modifica os parâmetros da sub-rotina em LM através de POKEs.

Ao terminar a transferência, um OUT na porta &HA8 (porta A do 8255A) deixa as quatro paginas do slot 2 ativas (RAM integral no sistema). De fato, o número 170 (&HAA) corresponde em binário a &B10101010, fazendo com que as quatro páginas estejam ativas no slot 2 (&B10), como mostra a figura 3.6 . No Hot-Bit as quatro páginas de RAM estão no slot 3.

fig. 3.6 - Exemplo de configuração de memória no Expert.

Exemplificando uma aplicação de alteração do interpretador, mostraremos em seguida como alterar as mensagens de erro enviadas pelo interpretador BASIC do Expert, passando-as para o português.

Após rodar o COPMEM, basta fazer POKEs nas regiões da RAM (correspondentes à ROM) onde estão guardadas as mensagens. O programa da figura 3.7 faz exatamente isso.

Usando um raciocínio análogo ao do COPMEM, vamos agora copiar o conteúdo da VRAM (memória de vídeo) para a página 1 do slot 2 (no Expert) ou do slot 3 (no Hot-Bit). Obviamente, é preciso ressetar o computador, pois nesse momento o interpretador está ativo na RAM e a cópia da tela o destruiria. A página 1 foi escolhida porque na página 0 da ROM residem as rotinas do BIOS que precisam estar ativas para essa transferência.

Basicamente, a operação do programa listado na figura 3.8 consiste no chaveamento da página 1 do slot 0 para o slot 2 (no Expert) ou

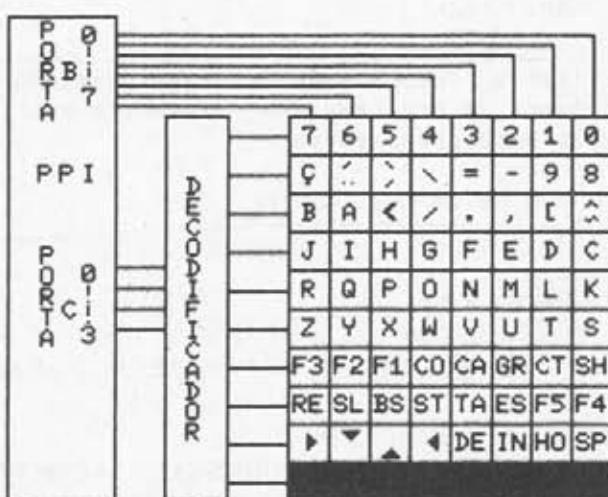
para o slot 3 (no Hot-Bit) e, através do BIOS, copiar o conteúdo da VRAM nessa página. No final, a página 1 é ativada novamente no slot 0. Reciprocamente, pode-se copiar o conteúdo dessa página na VRAM. Os endereços dessas rotinas no BIOS são, respectivamente, &H0059 e &H005C .

fig. 3.7 - Mensagens de erros (para o Expert).

```
10  REM Mensagens de Erro em Portugues
20 DATA NEXT sem FOR,Erro de Sintaxe,RE
TURN sem GOSUB,Fim de Dados,Funcao Ileg
al,Overflow,Falta Memoria,Linha Indefin
ida,Subscritor Invalido,Array Redimensi
onado
30 DATA Divisao por Zero,Direto Ilegal,
Tipo Invalido,Falta Espaco String,Strin
g Muito Longa,Formula String Complexa,I
mpossivel,Funcao Indefinida,Erro de E/S
,Nao Gravou,Falta RESUME,RESUME Sem Err
o,Nao Imprimivel,Falta Operando
40 DATA Excedeu Buffer,Excedeu Campo,Er
ro Interno,Numero Arq Invalido,Nao Enco
ntrado,Arq Ja Aberto,INPUT Apos fim,Nom
e Arq Invalido,Comando Direto no Arq,So
 E/S Sequencial,Arq nao Aberto,Nao Impr
imivel
50 DATA 16339,na,16348,Parou,19258,?Ree
ntre dados  ,19503,Extra ignorado,28927
,"Lendo:",28934,"Pulou:",0,FIM
60 CLS:E=15734:FOR I=1 TO 36:READ A$:PR
INTA$
70 FOR J=1 TO LEN(A$):POKE E,ASC(MID$(A
$,J)):E=E+1:NEXT J:POKE E,0:E=E+1:NEXT
I
80 READ E,A$:PRINTA$:IF A$="FIM" THEN E
ND ELSE FOR J=1 TO LEN(A$):POKE E+J-1,A
SC(MID$(A$,J)):NEXT J:GOTO 80
90 REM Usar apos o COPMEM
100 REM Nao usar COPMEM com Basic Disco
```

fig. 3.8 - Cópia da VRAM na RAM e vice-versa.

```
10 DATA F3,3E,A8,D3,A8,21,00,00,11,00,4
0,01,00,40,CD,59,00,3E,A0,D3,A8,FB,C9
20 DATA F3,3E,A8,D3,A8,21,00,40,11,00,0
0,01,00,40,CD,5C,00,3E,A0,D3,A8,FB,C9
30 SCREEN 0:WIDTH 38
```

```
40 FOR L=0 TO 45:READ A$:POKE 60000!+L,
VAL("&H"+A$):NEXT L:IF PEEK(32513)<>71
THEN POKE 60002!,255:POKE 60018!,240:PO
KE 60025!,255:POKE 60041!,240
50 DEF USR0=60000!:DEF USR1=60023!
60 REM F1 armazena uma tela
70 REM F2 recupera a tela salva por F1
80 KEY (1) ON:KEY (2) ON:ON KEY GOSUB 1
00,110
90 GOTO 120
100 X=USR0(0):RETURN
110 X=USR1(0):RETURN
120 REM Aqui Inicio do seu programa
130 SCREEN 2:FOR N=0 TO 255:PSET(N,90-9
0*SIN(N/128*3.14)):NEXT N
140 GOTO 140
150 SCREEN 2:X=USR1(0)
160 GOTO 160
```

Controle do teclado

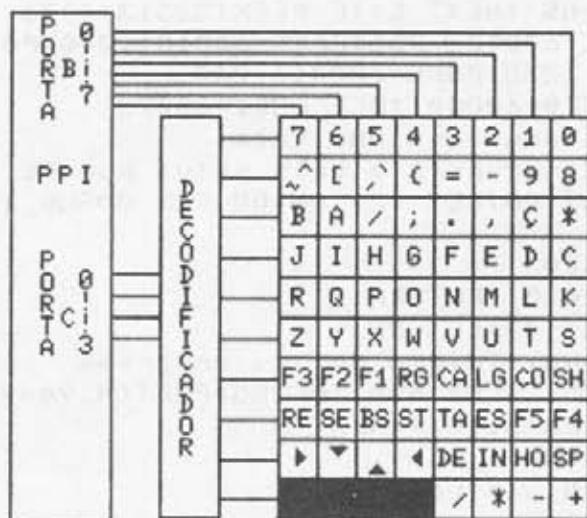
A porta B e a metade inferior (bits 0,1,2 e 3) da porta C são responsáveis pela varredura do teclado.

A parte baixa da porta C, de PC0 a PC3, conta de 0 a 9, ativando seqüencialmente cada uma das dez linhas da matriz do teclado (fig. 3.9). Durante a ativação de cada linha, a porta B lê quais colunas da matriz estão ativas, identificando desse modo quais teclas estão pressionadas.

fig. 3.9a - Matriz do teclado no HOTBIT (versão 1.1).

fig. 3.9b - Matriz do teclado no EXPERT (versão 1.1).

Você pode ler qualquer tecla através do BASIC com as funções INP e OUT. Para isso, basta colocar na parte baixa da porta C (bits de 0 a 3) o número da linha que deve ser verificada (de 0 a 9) e ler pela porta B se há alguma tecla pressionada.

No programa da figura 3.10 encontra-se um exemplo para saber se a tecla LGRA ou GRAPH está pressionada.

fig. 3.10 - Exemplo de uso do IN e do OUT.

```
1000 PC=INP(&HAA) AND &HF0 OR &H6
1010 OUT &HAA,PC
1020 X=INP(&HA9) AND &B00000100
1030 IF X THEN PRINT "NAO APERTADA" ELS
E PRINT "APERTADA"
1040 GOTO 1000
```

Para facilitar a compreensão do funcionamento do teclado, na figura 3.11 apresentamos um programa que desenha a matriz do teclado e aguarda que uma tecla seja pressionada.

fig. 3.11a - Programa Matriz do Teclado para o Expert.

```
10 ' PROGRAMA PPI
20 OPEN "GRP:" FOR OUTPUT AS#1
30 DEFINT A-Z:COLOR 15,4,4:SCREEN 2,1,0:
BEEP
40 ' DEFINE UM QUADRADO
50 SPRITE$(0)=CHR$(255)+CHR$(129)+CHR$(1
29)+CHR$(129)+CHR$(129)+CHR$(129)+CHR$(1
29)+CHR$(255)
60 ' DESENHA PPI E DECODER
```

```
70 LINE(16,0)-(55,191),15,B:BEEP:BEEP:LI
NE(74,39)-(97,189),15,B:BEEP:BEEP
80 ' DESENHA O TECLADO
90 FOR F=39 TO 189 STEP 15:LINE (120,F)-
(240,F):BEEP:NEXT F:FOR F=120 TO 240 STE
P 15:LINE(F,39)-(F,189):BEEP:NEXT F
100 ' LIGACAO PPI-DECODER
     LIGACAO DECODER-TECLADO
110 FOR F=98TO122STEP8:LINE(55,F)-(74,F)
,15:BEEP:NEXTF:FOR F=47TO189STEP15:LINE(
97,F)-(120,F),15:BEEP:NEXTF
120 ' LIGACAQ TECLADO-PPI
130 E=0:FORF=35TO7STEP-4:LINE(55,F)-((12
8+E),F),15:E=E+15:BEEP:NEXTF:E=7:FORF=23
3TO128STEP-15:LINE(F,E)-(F,39),15:BEEP:E
=E+4:NEXTF
140 ' COLOCA LETRAS NO TECLADO
150 FOR E=43 TO 132 STEP 15:FOR F=125 TO
 240 STEP 15:PRESET(F,E):READ A$:PRINT#1
,A$:BEEP:NEXT F,E
160 ' COLOCA FUNCOES NA MATRIZ
170 FOR E=133 TO 179 STEP 15:FOR F=122 T
O 237 STEP 15:PRESET(F,E):READ A$,B$:PRI
NT#1,A$:PRESET((F+6),E):PRINT#1,B$:BEEP:
NEXT F,E
180 ' COLOCA ACENTOS
190 PRESET(128,57):PRINT#1,CHR$(94):PRES
ET(123,64):PRINT#1,CHR$(126):PRESET(159,
57):PRINT#1,CHR$(96):PRESET(153,64):PRIN
T#1,CHR$(39)
```

```
200 ' COLOCA CURSOR
210 PRESET(127,163):PRINT#1,CHR$(&HCF):P
RESET(140,162):PRINT#1,CHR$(&HCD):PRESET
(155,165):PRINT#1,CHR$(&HCE):PRESET(168,
163):PRINT#1,CHR$(&HD0)
220 ' PINTA QUADRADOS NAO USADOS
230 FOR F=130TO175STEP15:PAINT(F,180),15
,15:NEXT F
240 ' COLOCA LEGENDA PPI DECOD
250 FOR F=2TO40STEP8:READ A$:PRESET(28,F
):PRINT#1,A$:PRESET(28,(F+88)):PRINT#1,A
$:BEEP:NEXTF
260 READA$:PRESET(36,18):PRINT#1,A$:READ
 A$:PRESET(36,111):PRINT#1,A$
270 FORF=4TO28STEP8:READA$:PRESET(43,F):
PRINT#1,A$:READA$:PRESET(43,(F+93)):PRIN
T#1,A$:NEXT F:PRESET(27,58):PRINT#1,"PPI
"
280 FORF=63TO164STEP8:READA$:PRESET(82,F
):PRINT#1,A$:NEXTF:CLOSE#1
290 ' ATUALIZA A VARREDURA
300 IF C=10THENC=0ELSEC=C+1
310 ' FAZ VARREDURA E LE TECLA
```

```
320 A=INP(170):B=(240ANDA)ORC:OUT170,B:D
=INP(169):IFD=255THEN 300
330 IF D=254THENX=0
340 IF D=253THENX=1
350 IF D=251THENX=2
360 IF D=247THENX=3
370 IF D=239THENX=4
380 IF D=223THENX=5
390 IF D=191THENX=6
400 IF D=127THENX=7
410 ' POSICIONA SPRITE
420 PUTSPRITE5,((225-X*15),(C*15+38)),10
,0:BEEP:BEEP:BEEP:GOTO300
430 DATA 7,6,5,4,3,2,1,0,,[,,(,=,-,9,8,B
,A,/,;,.,",",,*,J,I,H,G,F,E,D,C,R,Q,P,O,
N,M,L,K,Z,Y,X,W,V,U,T,S
440 DATA F,3,F,2,F,1,R,G,C,A,L,G,C,O,S,H
,R,E,S,E,B,S,S,T,T,A,E,S,F,5,F,4,,,,,,,,
,D,E,I,N,H,O,S,P,N,N,N,N,N,N,N,N,,/,,*,,
-,,+
450 DATA P,O,R,T,A,B,C,0,0,!,!,!,!,7,3,D
,E,C,O,D,I,F,I,C,A,D,O,R
```

fig. 3.11b - Alterações para que o programa anterior rode no Hot-Bit.

```
160 'COLOCA FUNCOES NA MATRIZ
170 FOR E=133 TO 164 STEP 15:FOR F=122
TO 237 STEP 15:PRESET(F,E):READ A$,B$:P
RINT#1,A$:PRESET((F+6),E):PRINT#1,B$:BE
EP:NEXT F,E
180 'COLOCA ACENTOS
190 PSET(140,65):PSET(143,65):PRESET(13
9,57):PRINT#1,CHR$(39):PRESET(155,64):P
RINT#1,"'":PRESET(155,57):PRINT#1,CHR$(
96):PRESET(230,72):PRINT#1,CHR$(94):PRE
SET(230,78):PRINT#1,CHR$(187)
220 'PINTA QUADRADOS NAO USADOS
230 FOR F=130TO235STEP15:PAINT(F,180),1
5:NEXT F
430 DATA 7,6,5,4,3,2,1,0,~,,,\,=,-,9,8,
B,A,<,/,.,",",[,,J,I,H,G,F,E,D,C,R,Q,P,
O,N,M,L,K,Z,Y,X,W,V,U,T,S
440 DATA F,3,F,2,F,1,C,O,C,A,G,R,C,T,S,
H,R,E,S,L,B,S,S,T,T,A,E,S,F,5,F,4,,,,,,
,,,D,E,I,N,H,O,S,P
```

Ao se pressionar uma tecla (inclusive SHIFT, CONTROL, LGRA ou GRAPH, RGRA ou CODE, etc.), o computador acende o quadrado correspondente da matriz, dando uma indicação precisa da localização da tecla.

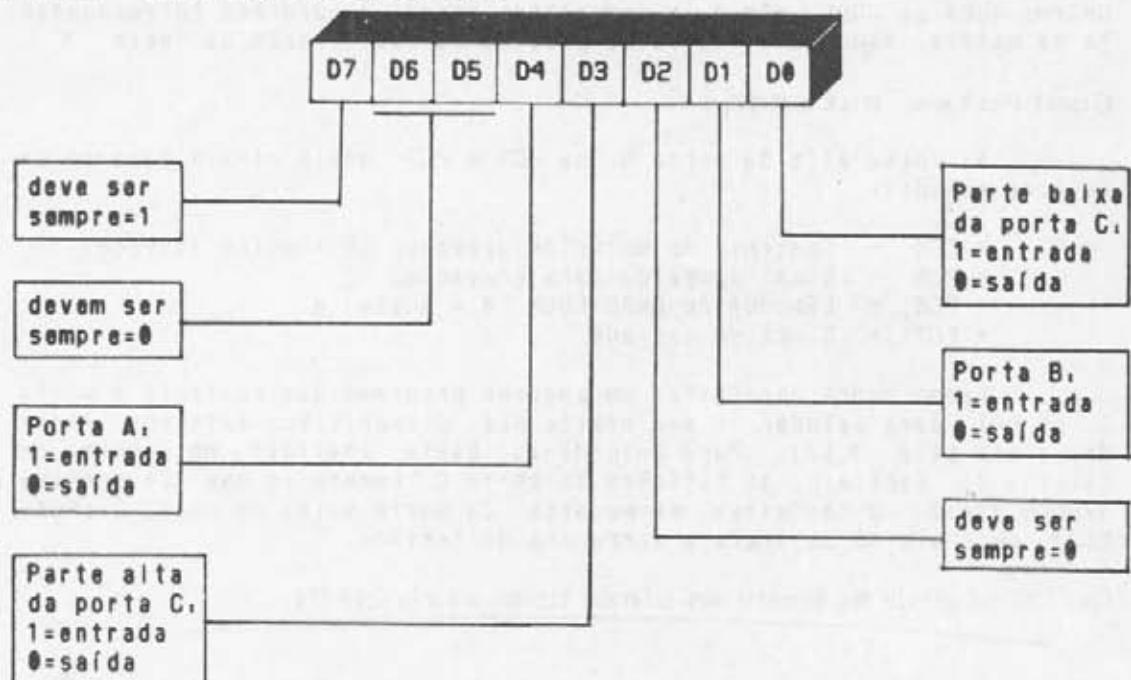
Controles externos

A parte alta da porta C, de PC4 a PC7, envia sinais para as seguintes funções:

* PC4 - Controle do motor do gravador (0 = motor ativado);
* PC5 - Sinal de saída para gravação;
* PC6 - Lâmpada do CAPS LOCK (0 = acesa) e
* PC7 - Click do teclado.

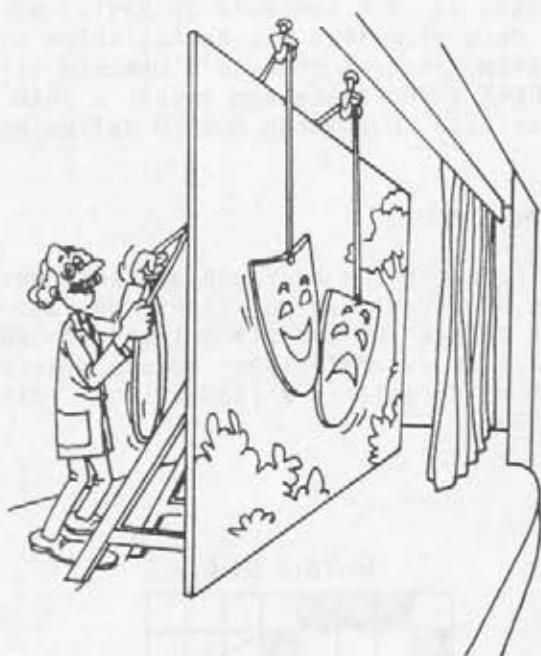
Vamos agora apresentar um pequeno programa que controla a porta C da PPI, para estudar o seu efeito nos dispositivos externos ligados a ela (fig. 3.12). Para entendê-lo, basta analisar no esquema do início do capítulo, as ligações da porta C. Lembre-se que não surtirá nenhum efeito a tentativa de mudança da parte baixa da porta C (PC0-PC3), pois ela só controla a varredura do teclado.

fig. 3.12 - Controle dos dispositivos externos ligados a porta C da PPI.

```
10 REM Controle da PPI/Porta 170
20 CLS:PRINT "Controle PPI"
30 PRINT:INPUT "Numero do bit (4-7) ";B
40 INPUT "Set ou Reset (S/R) ";A$
50 B$=BIN$(INP(170)):B$=STRING$(8-LEN(B
$),48)+B$
60 PRINT"Porta 170:";B$;" (antes)":PRIN
T TAB(10);"76543210"
70 D$="0":IF A$="S" OR A$="s" THEN D$="
1"
80 MID$(B$,8-B)=D$:PRINT TAB(10);B$;" (
depois)
90 OUT 170,VAL("&B"+B$):I$=INPUT$(1):RU
N
```

A porta de controle da PPI

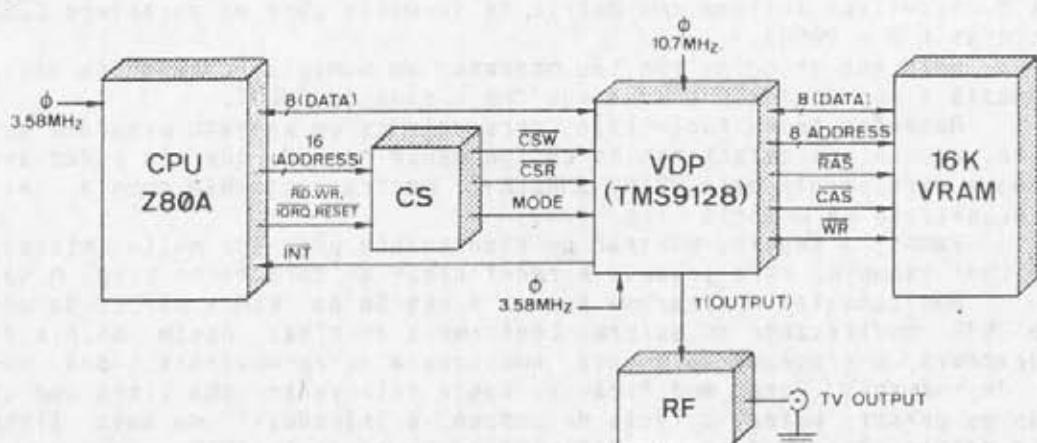
Apenas para efeito de ilustração, vamos apresentar a estrutura da porta D da PPI (porta específica para controle).

Essa porta é a que define o funcionamento das portas A, B e C (fig 3.13).

fig. 3.13 - Estrutura da porta de controle.

Para se ter a porta A como saída, a porta B como entrada e a porta C como saída(tanto a parte baixa quanto a parte alta) é necessário enviar o número &B10000010 (&H82 , em hexadecimal) para a porta de controle. Isso pode ser feito através do comando:

```
OUT &HAB,&H82
```

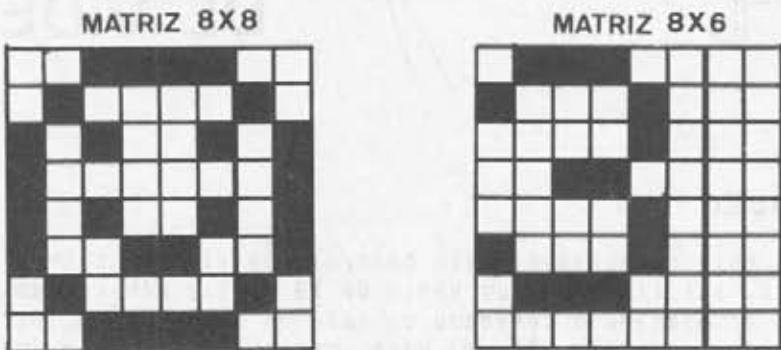
Esse é um dos primeiros comandos executados pelo micro assim que ele é ligado.

Não tente enviar outros valores para a porta de controle para não correr o risco de programar as portas B e C como saídas. Isso pode acabar danificando o circuito do teclado ou o próprio teclado, pois ao pressionar uma tecla você estará pondo em curto-circuito duas saídas!

Capítulo 4

VDP-INTERFACE DE VÍDEO

Introdução

O chip responsável pelo controle do vídeo é o TMS9128A da Texas Instruments, auxiliado por um banco de 16 Kbytes RAM (denominado VRAM), onde fica armazenado o conteúdo da tela de imagem. O HOTBIT possui uma saída de vídeo composto (PAL-M) para monitor a cores e uma saída de RF para o canal 3 ou 4 de uma TV comum. O Expert em sua segunda versão tem, além disso, uma saída para vídeo monocromático e outra para monitor RGB. Em sua primeira versão, a saída de RF e vídeo composto eram obtidas através do adaptador TA-1 que vinha junto com a CPU.

fig. 4.1 - Diagrama das ligações do VDP.

A VRAM (Video RAM) é um banco de 16 Kbytes independente da memória acessada diretamente pela CPU. Devido ao fato de seu acesso ser feito exclusivamente através da VDP, qualquer endereçamento entre 0 e 64 Kbytes pode ser usado sem perigo de conflito. No MSX, o VDP endereça a VRAM entre 0 e 16383 .

Os comandos básicos de operação do VDP (através do Basic) são VPEEK, VPOKE, BASE, VDP E SCREEN. Os dois primeiros são equivalentes ao PEEK e POKE normais. Esses últimos, porém, acessam somente a memória ligada diretamente à CPU, enquanto VPEEK e VPOKE acessam apenas a VRAM. BASE e VDP serão vistos no final do capítulo. O comando SCREEN define os vários modos de operação do VDP.

SCREEN 0 (modo texto, 40 x 24)

Neste modo de operação, cada caractere é representado por uma matriz de 8 x 6 para possibilitar os 40 caracteres por linha. De fato, experimente digitar VPOKE 30,1 e você obterá um caractere truncado no fim da primeira linha da tela, pois eles sao definidos em uma matriz 8 x 8. Obviamente, isto não ocorre com os caracteres alfanuméricos, pois eles não ocupam toda a matriz (fig. 4.2).

fig. 4.2 - Caracteres 8x8 e 8x6.

Com o VPOKE do exemplo acima, quisemos mostrar também a organização da VRAM para a SCREEN 0, que é a seguinte:

0 a 959: posições dos caracteres na tela (40x24=960);
2048 a 4095: tabela de formação dos caracteres, sendo que cada 8 bytes consecutivos definem uma matriz de formação para um caractere (256 caracteres x 8 = 2048).

Note que as cores não são mapeadas em memória, podendo-se definir apenas a cor do texto e do fundo com o comando COLOR.

Baseados nesse raciocínio, apresentamos um pequeno programa que permite acesso aos caracteres de código menor que 32 (que não podem ser impressos diretamente pelo PRINT CHR$(N)), mostrando também como a tela está organizada na memória (fig. 4.3).

Vamos, a seguir, mostrar um recurso que pode ser muito interessante (por exemplo, para jogos): a redefinição de caracteres (fig. 4.4).

Basicamente, o programa acessa a região na VRAM a partir da memória 2048, modificando os valores conforme o desejado. Assim, escolhido um caractere, o programa apresenta uma janela correspondente à sua matriz de formação. Para modificá-la, basta selecionar uma linha com as teclas de cursor, bater a tecla de espaço e introduzir uma nova linha para a matriz. Terminadas as modificações, basta teclar "F".

fig. 4.3 - Demonstração da estrutura da VRAM na SCREEN 0.

```
10 SCREEN 0
20 A$="***** EDITORA ALEPH *****"
30 FOR L=1 TO LEN(A$)
40 VPOKE L,ASC(MID$(A$,L))
50 NEXT L
60 FOR L=1 TO 31
70 VPOKE 2*L+LEN(A$),L
80 NEXT L
90 LOCATE 0,4
100 END
```

fig. 4.4 - Redefinidor de caracteres na SCREEN 0.

```
10 REM REDEFINIDOR - SCREEN 0
20 SCREEN 0:WIDTH 38
30 CLS:PRINT "Redefinidor de Caracteres
/0":PRINT
40 PRINT STRING$(8,219):FOR I=1 TO 8:PR
INT CHR$(219);:FOR J=1 TO 6:PRINT CHR$(
1);"G";:NEXT J:PRINT CHR$(219):NEXT I:P
RINT STRING$(8,219)
50 PRINT:PRINT:INPUT "Caracter ";A$:IF
LEN(A$)=2 AND LEFT$(A$,1)=CHR$(1) THEN
I=8*(ASC(RIGHT$(A$,1))-64)+2048:GOTO 70
60 I=8*ASC(A$)+2048
70 LOCATE 0,3:FOR J=I TO I+7:B$=BIN$(VP
EEK(J)\4):B$=STRING$(6-LEN(B$),48)+B$:P
RINT CHR$(219);
80 FOR K=1 TO 6:IF MID$(B$,K,1)="1" THE
N PRINT CHR$(219); ELSE PRINT CHR$(1);"
G";
90 NEXT K:PRINT:NEXT J:X=0
100 LOCATE 0,X+3:PRINT ">";CHR$(8);
110 J$=INKEY$:IF J$=CHR$(30) THEN PRINT
 CHR$(219):X=X-1:IF X<0 THEN X=7
120 IF J$=CHR$(31) THEN PRINT CHR$(219)
:X=X+1:IF X>7 THEN X=0
130 IF J$=" "THEN 150 ELSE IF J$="F" TH
EN 30
140 GOTO 100
150 LOCATE 0,20:PRINT"               ":LOC
ATE 0,20:INPUT L$:IF LEN(L$)<>6THEN 150
 ELSE VPOKE I+X,4*VAL("&B"+L$):LOCATE 1
,X+3:FOR L=1 TO 6:IF MID$(L$,L,1)="1" T
HEN PRINT CHR$(219);ELSE PRINT"G";
160 NEXT L
170 GOTO 100
Ok
■
```

Apesar do **padrão** MSX objetivar uma compatibilidade total entre os computadores de diversas marcas, isto **não** ocorre na realidade. Por exemplo, diferentes marcas usam diferentes tabelas de formação de caracteres (reveja as figuras 0.3 e 0.4). O programa da figura 4.4 pode ser usado para solucionar esse problema.

Uma outra aplicação interessante deste modo é usar a região não utilizada da VRAM para guardar telas. O programa da figura 4.5 faz exatamente isso.

fig. 4.5 - Guardando telas.

```
10 'PROGRAMA ARQUIVO DE TELAS DE TEXTO
20 KEYOFF:BEEP:COLOR 15,4:BASE(0)=0:SCR
EEN 0
30 FOR F=1TO7:KEY(F)ON:NEXTF
40 PRINT"   PARA ARQUIVAR AS TELAS,PRES
SIONE UMADAS FUNCTION KEY'S (1-5) PARA
TRABALHARCOM A TELA CORRESPONDENTE."
50 PRINT:PRINT:PRINT"   ESCREVA ALGUMA
COISA NAS TELAS E LO_GO APOS PRESSIONE
RETURN PARA GRAVAR A SUA TELA."
```

```
60 PRINT:PRINT:PRINT"   DEPOIS QUE VOCE
 DEFINIR AS TELAS DI_GITE F6 PARA VOLTA
R A ESTA TELA OU F7 ELOGO APOS A FUNCTI
ON KEY CORRESPONDENTEPARA VER A TELA QU
E VOCE ARMAZENOU."
70 ON KEY GOSUB 100,110,120,130,140,150
,160:GOTO 70
80 SCREEN 0:LOCATE 0,16:LOCATE0,16:PRIN
T"TELA";TELA:INPUT A$:CLS
90 LOCATE 0,0:PRINT A$:A$="":GOTO 70
100 BASE(0)=1024:TELA=1:RETURN 80
110 BASE(0)=4096:TELA=2:RETURN 80
120 BASE(0)=5120:TELA=3:RETURN 80
130 BASE(0)=6144:TELA=4:RETURN 80
140 BASE(0)=7168:TELA=5:RETURN 80
150 BASE(0)=0:RETURN 10
160 ON KEY GOSUB 170,180,190,200,210,22
0,230:GOTO 160
170 BASE(0)=1024:RETURN 160
180 BASE(0)=4096:RETURN 160
190 BASE(0)=5120:RETURN 160
200 BASE(0)=6144:RETURN 160
210 BASE(0)=7168:RETURN 160
220 BASE(0)=0:RETURN 70
230 RETURN
```

O programa de arquivo de telas, após fazer uma introdução, es-

pera o pressionamento de uma tecla de função (F1 a F5) para que ele possa selecionar uma das regiões da VRAM que ele irá usar como tela de texto (neste caso, mudando o valor do "BASE", função esta que será vista com detalhes ainda neste capítulo). Após ter sido selecionada a região da VRAM, o programa informa em qual tela está funcionando e espera a entrada de uma mensagem que ficará impressa na tela. Esta, então, será arquivada na região selecionada.

Para arquivar uma tela (de 1 a 5), pressione a tecla de função correspondente (ex: F1=tela 1, F2=tela 2, etc). Se você quiser voltar ao menu principal, pressione a tecla F6 (shift+F1), e se você quiser rever uma ou mais telas, pressione F7, que passará o programa para o modo de leitura. Se neste modo for pressionada uma tecla de função (F1 a F5), o programa não mais arquivará telas, mas sim apresentará a tela correspondente, tenha sido ela arquivada ou não.

SCREEN 1 (modo texto, 32 x 24)

Neste modo de operação, os caracteres são mostrados integralmente (8 x 8) possibilitando, entretanto, apenas 32 caracteres por linha. A estrutura da VRAM é bastante diferente neste modo:

6144 a 6911: posição dos caracteres na tela;
0 a 2047: tabela de formação dos caracteres;
8192 a 8223: tabela de cores dos grupos de 8 caracteres;

Com a tabela de cores, existe o recurso de se definir a cor de frente e de fundo de cada grupo de oito caracteres consecutivos em código, onde o primeiro do grupo deve ser múltiplo de 8. Acreditamos que o programa a seguir, quando rodado, seja bastante elucidativo com relação a definição das cores e à estrutura da VRAM (fig. 4.6).

fig. 4.6 - Estrutura da VRAM na SCREEN 1

```
10 COLOR 15,4,4:SCREEN 1
20 A$="***** EDITORA ALEPH *****"
30 FOR L=1 TO LEN(A$)
40 VPOKE L+6144,ASC(MID$(A$,L))
50 NEXT L
60 FOR L=1 TO 1000:NEXT L
70 FOR L=8192 TO 8223
80 VPOKE L,255*RND(1)
90 NEXT L
100 FOR L=0 TO 255
110 VPOKE 6144+L+LEN(A$),L
120 NEXT L
130 FOR L=1 TO 3000:NEXT L
140 GOTO 10
```

Lembre-se que, além das definições de cores mostradas acima, podemos também definir a cor da borda neste modo.

Seguindo a mesma linha de raciocínio usada na SCREEN 0, apresentamos na figura 4.7 um programa para redefinir caracteres na SCREEN 1.

Para quem quiser estudar uma aplicação interessante do SCREEN 1, aconselhamos a leitura do capítulo TANK do livro COLEÇÃO DE PROGRAMAS PARA MSX - Vol. 1 desta mesma editora.

fig. 4.7 - Redefinidor de caracteres na SCREEN 1.

```
10 REM REDEFINIDOR - SCREEN 1
20 SCREEN 1:WIDTH 28
30 CLS:PRINT "Redefinidor de Caracteres
/1"
40 PRINT
50 PRINT STRING$(10,219):FOR I=1 TO 8:P
RINT CHR$(219);:FOR J=1 TO 8:PRINT CHR$
(1);"G";:NEXT J:PRINT CHR$(219):NEXT I:
PRINT STRING$(10,219)
60 PRINT:PRINT:INPUT "Caracter ";A$:IF
LEN(A$)=2 AND LEFT$(A$,1)=CHR$(1) THEN
I=8*(ASC(RIGHT$(A$,1))-64):GOTO 80
70 I=8*ASC(A$)
80 LOCATE 0,3:FOR J=I TO I+7:B$=BIN$(VP
EEK(J)):B$=STRING$(8-LEN(B$),48)+B$:PRI
NT CHR$(219);
90 FOR K=1 TO 8:IF MID$(B$,K,1)="1" THE
N PRINT CHR$(219); ELSE PRINT CHR$(1);"
G";
100 NEXT K:PRINT:NEXT J:X=0
110 LOCATE 0,X+3:PRINT ">";CHR$(8);
120 J$=INKEY$:IF J$=CHR$(30) THEN PRINT
 CHR$(219):X=X-1:IF X<0 THEN X=7
130 IF J$=CHR$(31) THEN PRINT CHR$(219)
:X=X+1:IF X>7 THEN X=0
140 IF J$=" "THEN 160 ELSE IF J$="F" TH
EN 30
150 GOTO 110
160 LOCATE 0,20:PRINT"            ":LOC
ATE 0,20:INPUT L$:IF LEN(L$)<>8THEN 160
 ELSE VPOKE I+X,VAL("&B"+L$):LOCATE 1,X
+3:FOR L=1 TO 8:IF MID$(L$,L,1)="1" THE
N PRINT CHR$(219); ELSE PRINT CHR$(1);"
G";
170 NEXT L
180 GOTO 110
```

Para finalizar este modo de operação, seria necessário também analisar os sprites. Isso será feito mais adiante pois o procedimento de uso é análogo para as SCREENs 1, 2 e 3.

SCREEN 2 (alta resolução gráfica, 256 x 192)

Analogamente à SCREEN 1, na SCREEN 2 os caracteres são apresentados em sua forma integral (8 x 8). Além disso é possível o acesso ponto a ponto da tela para recursos gráficos. A estrutura da VRAM é também

semelhante ao modo 1:

6144 a 6911: posição dos caracteres da tela;
0 a 6143: três tabelas de formação de caracteres;
8192 a 14335: tabela de cores para cada grupo de 8 pontos.

Pode-se notar que a tabela de formação de caracteres é três vezes maior que a do modo 1, isto para possibilitar a definição de um caractere diferente para cada área 8 x 8 da tela. Isso não seria possível usando apenas os 256 caracteres disponíveis na ROM. Além disso, a tabela de cores é também muito maior, pois possibilita a definição de uma cor (de frente e fundo) para cada grupo de 8 pontos horizontais, fazendo com que, por exemplo, um único caractere 8x8 possa ter 8 cores diferentes de frente e de fundo. Todas essas coisas ocorrem em decorrência da possibilidade de acessar a tela para recursos gráficos.

Inicialmente, as tabelas de formação estão "vazias", justamente esperando a utilização gráfica. Portanto, para poder usar os caracteres é preciso carregá-los com os códigos originais da ROM. Além disso, a tabela de cores também está "vazia" (isto é, cor de frente igual a cor de fundo), sendo preciso também modificá-la. Acreditamos que o programa da figura 4.8 esclareça esta situação.

fig. 4.8 - Estrutura da VRAM na SCREEN 2.

```
10 COLOR 15,4,4:SCREEN 2
20 FOR K=0 TO 4096 STEP 2048
30 X=255*RND(2)
40 FOR L=0 TO 2047
50 VPOKE L+K,PEEK(L+7103):' 7103=ENDERE
CO INICIAL DA TABELA DE FORMACAO DE CAR
ACTERES NA ROM
60 VPOKE L+K+8192,X
70 NEXT L,K
80 FOR L=1 TO 2500:NEXT L
90 GOTO 20
```

Depois de digitá-lo e executá-lo, experimente retirar a linha 30 e modificar a linha 60 para:

```
60 VPOKE L+K+8192,255*RND(2)
```

Após as alterações, rode-o novamente e tente explicar o que ocorre. A seguir, tente fazer com que as isócromas sejam linhas horizontais de pontos (ou seja, faça com que cada linha horizontal da tela tenha uma só cor diferente das vizinhas)... Que tal fazer isócromas verticais ?

Para complicar mais ainda as coisas, vamos mostrar uma outra peculiaridade de comportamento da SCREEN 2: ao redefinir um "pixel" de um caractere na tela, os pontos acesos da linha horizontal correspondente do mesmo mudam acompanhando a cor do pixel (fig. 4.9).

Vamos mostrar agora um truque que permite escrever textos em SCREEN 2 sem ter que carregar a tabela de formação de caracteres contida na ROM. No MSX cada periférico é considerado um arquivo, incluindo a tela gráfica, cuja denominação para arquivo é "GRP:". Para apagar o que foi escrito é necessário outro truque: desenhar sobre a região a ser apagada um retângulo "cheio" da cor do fundo (com o comando LINE/BF). Isso acontece porque o ato de escrever uma nova mensagem na mesma posição não apaga a anterior. Na figura 4.10 apresentamos um programa que ilustra isso.

fig. 4.9 - Efeito do PSET nas cores de "pixels" vizinhos.

```
10 COLOR 15,1,1
20 SCREEN 2
30 LINE (132,90)-(176,180),7,BF:FOR T=1
TO 1000:NEXT T
40 FOR L=1 TO 15
50 FOR M=70 TO 200
60 PSET (128,M),L
70 PSET (180,M),L
80 NEXT M
90 NEXT L
100 GOTO 40
```

fig. 4.10 - Escrevendo textos na SCREEN 2.

```
10 REM Este programa escreve na tela de
       alta resolucao (SCREEN 2)
20 OPEN "GRP:" FOR OUTPUT AS 1
30 REM Texto para a impressao
40 A$="Ola, eu sou seu micro MSX"
50 COLOR 1,15,15:SCREEN 2
60 REM Figuras em alta resolucao
70 FOR I=0 TO 250 STEP 10:LINE(I,0)-(I,
190),8:NEXT I:FOR I=0 TO 190 STEP 10:LI
NE(0,I)-(250,I),8:NEXT I
80 REM Posicao de inicio da impressao
90 PRESET (24,88)
100 REM Imprimir
110 PRINT #1,A$
120 FOR I=1 TO 400:NEXT I
130 REM Apagar
140 LINE (16,88)-(230,95),15,BF
150 FOR I=1 TO 200:NEXT I
160 GOTO 90
```

O programa da figura 4.10 sugere um modo razoavelmente fácil de se colocar pequenos textos explicativos numa figura já desenhada. No entanto, a combinação "indiscriminada" de texto e gráfico não é muito simples, pois uma linha de gráfico que passe perto de uma parte de texto pode alterá-la sensivelmente. Para podermos ter, numa mesma tela, gráfico e texto, sendo este último manipulável de maneira semelhante à da SCREEN 1, poderemos então reservar um terço da tela apenas para o texto. Note que entre as vantagens de manipular o texto deste modo estão a velocidade de impressão e a facilidade de apagamento (bastando dar o VPOKE do código 32 - espaço). Na figura 4.11 apresentamos um programa que reserva parte da SCREEN 2 só para textos.

Uma aplicação interessante do conhecimento da estrutura da SCREEN 2 é a possibilidade de se copiar a imagem gráfica da tela na impressora, desde que esta tenha recursos gráficos. Na verdade, o único trabalho a ser feito seria pesquisar a tabela de padrões da tela gráfica, comandar a impressora para entrar no modo gráfico e enviar os códi-

gos lidos nesta tabela.

Entretanto, existe um problema que decorre do funcionamento do comando PAINT. Quando o byte da tela a ser "pintado" é não nulo, o PAINT "acende" os pontos restantes deste byte e muda sua cor de frente para a cor do PAINT. Mas, se o conteúdo do byte é zero, em vez de os bits serem "acesos", eles permanecem apagados e quem recebe a cor do PAINT é o atributo da cor de fundo.

Na tela, isto não causa nenhum problema, pois os pontos estão apagados, mas a cor de fundo está igual a cor do PAINT e por isso a figura parece estar totalmente pintada na tela. Só que na hora de pesquisar a tabela de padrões, alguns pontos estarão apagados (apesar de parecerem acesos na tela), o que ocasiona o surgimento de alguns "buracos" na figura impressa.

Um programa com a solução para este problema é apresentado detalhadamente no capítulo 6.

fig. 4.11 - Transformando o terço inferior da SCREEN 2 em SCREEN 1.

```
10 REM Este programa combina texto com
       gráficos.
20 REM Inicialização
30 SCREEN 2
40 FOR I=0 TO 255: VPOKE I+6656,0: NEXT
 I
50 FOR I=0 TO 2047:VPOKE 4096+I,PEEK(71
03+I): VPOKE 12288+I,112: NEXT I
60 REM Figuras em alta resolução
70 FOR I=1 TO 50:LINE (RND(1)*255,RND(1
)*127)-(RND(1)*255,RND(1)*127):NEXT I
80 LINE (0,127)-(255,127)
90 REM Texto para imprimir
100 P=6656
110 FOR I=1 TO 8:READ A$
120 FOR J=1 TO LEN(A$)
130 VPOKE P+J,ASC(MID$(A$,J)):NEXT J
140 P=P+32:NEXT I
150 GOTO 150
160 DATA Olá !!!,Eu sou um micro MSX br
asileiro ,e estou aqui demonstrando com
o,e possível combinar textos com,gráfic
os de uma maneira dife-,rente da conven
cional.,Este é mais um pequeno truque,d
os programadores da Aleph.
```

Como fizemos na SCREEN 1, deixaremos a análise dos sprites mais para a frente.

SCREEN 3 (modo gráfico multicolor, 64 x 48)

Este é o modo de menor resolução do computador. Os caracteres também são mostrados integralmente (8 x 8), mas obviamente em tamanho maior que nos outros modos. A estrutura da VRAM é a seguinte:

2048 a 3583: Posição dos caracteres na tela;
0 a 2047: Tabela de cores dos caracteres.

É importante salientar que sendo a SCREEN 3 de pouca utilidade prática e sendo difícil a compreensão de seu funcionamento, vamos nos limitar basicamente a apresentar alguns programas práticos.

Começaremos por um programa que cria títulos para uso, por exemplo, em aberturas de filmes em videocassete (fig. 4.12).

fig. 4.12 - Titulador.

```
10 'Titulador de Vídeo-Cassete
20 'Título
30 READN:Y=96-16*N:DIMA$(N):FORI=1 TO N
:READ A$(I):D(I)=128-16*LEN(A$(I)):NEXT
I
40 'Inicialização
50 SCREEN 3: OPEN"grp:" FOR OUTPUT AS #
1
60 'Impressão
70 COLOR RND(1)*13+1
80 FOR I=1 TO N
90 PRESET(D(I),Y+32*I-32)
100 PRINT#1,A$(I)
110 NEXT I
120 GOTO 70
130 DATA 4,O beijo,da,Mulher,Aranha
```

Um outro exemplo interessante é consequência do pouco espaço que a SCREEN 3 exige da VRAM: pode-se armazenar telas nas regiões restantes. O programa da figura 4.13 gera três desenhos e os coloca na VRAM. Para chamá-los em sequência, basta apertar qualquer tecla.

fig. 4.13 - Guardando telas da SCREEN 3.

```
10 'Programa arquivo de várias telas de
 modo gráfico screen 3
20 BASE(18)=BASE(19)
30 BASE(17)=0:COLOR 15,4,4:SCREEN 3
40 FOR F=0TO255
50 Y=96+90*SIN(F/32*3.14)
60 PSET(F,Y),7
70 NEXT F
80 BASE(17)=4096:SCREEN 3
90 FOR F=0TO255
100 Y=96+90*COS(F/32*3.14)
110 PSET(F,Y),8
120 NEXT F
130 BASE(17)=6144:SCREEN 3
140 FOR F=0TO255
150 X=96+90*COS(F/64*3.14)
160 Y=96+90*SIN(F/16*3.14)
170 PSET(X,Y),15
180 NEXT F
190 BASE(17)=0:A$=INPUT$(1):BASE(17)=40
96:A$=INPUT$(1):BASE(17)=6144:A$=INPUT$
(1):A$="":GOTO 190
```

Note que a organização da SCREEN 3 é bastante complexa, não sendo seu estudo de efeito prático relevante. Se houver interesse em se aprofundar neste tema, consulte o manual técnico intitulado "TMS9118/TMS9128/TMS9129 Data Manual - Video Display Processors", que pode ser encontrado junto à Texas Instruments.

Para finalizar a SCREEN 3 (com exceção dos sprites), vamos apresentar algumas "brincadeiras"

fig. 4.14 - Tapete persa.

```
0 '######## TAPETE PERSA ############
20 COLOR 15,7,4:SCREEN 3
30 '#### DEFINE O LIMITE DA TELA ####
40 FOR F=0 TO 1537
50 '#### GERA UMA COR RANDOMICA #####
60 A=RND(1)*255
70 '##### COLOCA A COR NA TELA ######
80 VPOKE F,A
90 NEXT F
100 '### INDICA O FIM DOS VPOKE'S ####
110 BEEP:BEEP:BEEP:BEEP:BEEP
120 '# ESPERA TECLA SER PRESSIONADA ##
     ###### PARA REINICIAR ###########
130 A$=INPUT$(1):CLS
140 GOTO 20
```

fig. 4.15 - Galáxias I.

```
10 '-------------------------------------
              GALAXIAS I
   -------------------------------------
20 FOR Q= 1 TO 3:READTF,DT,D,WX,F,WY
30 COLOR7,1,1:SCREEN 3
40 FOR T=0TOTFSTEPDT
50 R=84*EXP(-D*T)
60 X=R*SIN(WX*T+F)
70 Y=R*COS(WY*T)
80 PSET((128+X),(96+Y)),7
90 NEXT T
100 NEXT Q:BEEP:BEEP:BEEP:BEEP:BEEP
110 GOTO110
120 DATA80,.2,.02,2,.2,2,80,.1,.01,2,.8
,2,80,.1,.015,2,0,6
```

fig. 4.16 - Galáxias II.

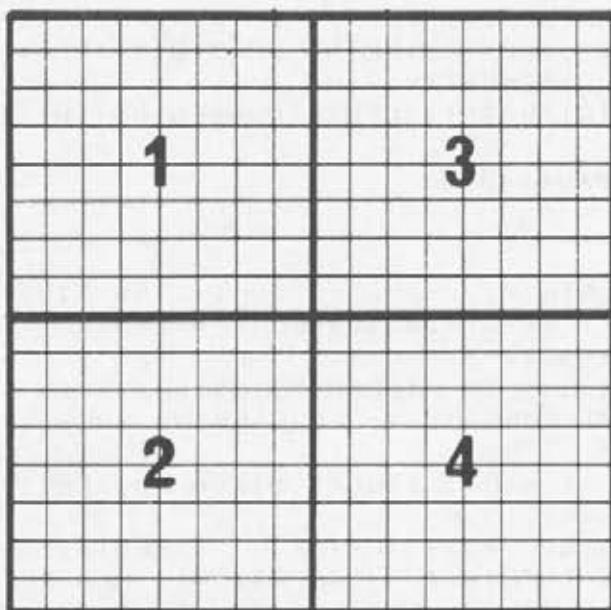
```
10 '_______________________________
              GALAXIAS II
   _______________________________
20 C=2
30 FOR Q=0 TO 1:READ TF,DT,D,WX,F,WY
40 COLORC,1,1:SCREEN 3
50 FOR T=0TOTFSTEPDT
60 R=84*EXP(-D*T)
70 X=R*SIN(WX*T+F)
80 Y=R*COS(WY*T)
90 PSET((128+X),(96+Y)),C
100 IFC=15THENC=2ELSEC=C+1
110 NEXT T
120 NEXT Q:BEEP:BEEP:BEEP:BEEP:BEEP
130 GOTO130
140 DATA 01,.1,.025,5,0,5,80,.1,.045,2,
7,2
Ok
■
```

Sprites

Um dos recursos mais característicos (e interessantes) do MSX são os sprites, disponíveis na SCREEN 1, SCREEN 2 e SCREEN 3. Como a operação deles é idêntica nos três modos, vamos comentá-los em separado.

Os sprites são figuras definíveis pelo usuário que podem ser superpostas à uma das SCREENs encobrindo a imagem normal (textos ou gráficos). Existem na VRAM 2048 bytes para a definição dos sprites. Se os sprites forem na forma 8 por 8, este espaço permite definir 256 padrões diferentes. Se forem 16 por 16, então podem ser definidos apenas 64 padrões.

Existem 32 níveis de sprites que podem ser usados ao mesmo tempo na tela. Cada um destes níveis tem um número que o identifica (0 a 31). Note que níveis de número menor têm prioridade de impressão sobre os de número maior (ou seja, estão num plano mais próximo do observador) e, efetivamente, figuras definidas num plano mais próximo tendem a encobrir ("tampar") figuras de planos mais distantes.

Para controlar estes 32 planos, existe uma tabela de atributos (de 128 bytes) dividida em 32 grupos de 4 bytes, onde cada grupo está associado a um plano. A divisão destes quatro bytes é:

0 - Coordenada Y do sprite (de 0 a 191)
1 - Coordenada X do sprite (de 0 a 255)
2 - Código da cor do sprite projetado (0 a 15)
3 - Número do padrão do desenho (0-255, em 8x8 e 0-63 em 16x16)

Vamos então analisar como se formam os sprites.

Os padrões dos sprites 8x8 são formados por sequências de oito bytes, da mesma forma que ocorre com os caracteres no modo SCREEN 1. A principal diferença é que os bits que contêm zeros não indicam pontos propriamente apagados, mas sim transparentes. Isto significa que se o sprite for colocado sobre uma imagem, os pontos apagados deixarão transparecer a porção da imagem correspondente aos mesmos. Outra diferença consiste no fato de que o sprite pode ser colocado em qualquer posição da SCREEN 1 , enquanto os caracteres só podem ser colocados em posições "estanques".

Um sprite 16x16 é formado por um "agrupamento" de quatro sprites 8x8 (ver figura 4.17), usando, portanto, 32 bytes para ser formado. Naturalmente esta noção de agrupamento é apenas visual, pois o seu controle lógico é feito pelo computador como sendo de um único sprite de tamanho maior.

fig. 4.17 - Sprites 16x16 .

SPRITE$(N)= 1 + 2 + 3 + 4

Criando sprites

A maneira mais fácil de se criar um sprite é usando a função do Basic:

SPRITE$(n)="sequência de caracteres"

onde n é o número do padrão definido (0-255 para 8x8, 0-63 para 16x16). A sequência contém os caracteres cujos códigos correspondem aos valores dos bytes que formam o padrão de cima para baixo (sprites 8x8 usam 8 bytes, sprites 16x16 usam 32 bytes).

A sequência de 32 bytes que compõe o sprite 16x16 nada mais é que a concatenação das quatro sequências de oito bytes que formam os quatro sprites menores, e esta concatenação deve seguir a ordem da figura 4.17 .

A seguir são apresentados dois programas: o editor de sprites 8x8 e o editor para 16x16. O funcionamento de ambos é quase idêntico. Primeiro deve-se fornecer o número do padrão do sprite desejado. Em seguida, com as teclas de cursor, seleciona-se a linha a alterar e, com a barra de espaço, procede-se a alteração pela qual deve-se entrar uma sequência de "1"s e "0"s (oito para 8x8 e dezesseis para 16x16). Ao terminar, deve-se digitar "F".

fig. 4.18 - Editor de sprite 8x8 .

```
10 REM Editor de sprites
20 SCREEN 1,1:COLOR 15,4,4:WIDTH 28
30 KEYOFF:CLS:PRINT "Redefinidor de Spr
ites 8x8"
40 PRINT
50 PRINT STRING$(10,219):FOR I=1 TO 8:P
RINT CHR$(219);:FOR J=1 TO 8:PRINT CHR$
(1);"G";:NEXT J:PRINT CHR$(219):NEXT I:
PRINT STRING$(10,219)
60 PRINT:PRINT:INPUT "Numero do sprite
";N
70 I=8*N+14336
```

```
80 LOCATE 0,3:FOR J=I TO I+7:B$=BIN$(VP
EEK(J)):B$=STRING$(8-LEN(B$),48)+B$:PRI
NT CHR$(219);
90 FOR K=1 TO 8:IF MID$(B$,K,1)="1" THE
N PRINT CHR$(219); ELSE PRINT CHR$(1);"
G";
100 NEXT K:PRINT:NEXT J:X=0:PUTSPRITE 0
,(140,40),15,N
110 LOCATE 0,X+3:PRINT ">";CHR$(8);
120 J$=INKEY$:IF J$=CHR$(30) THEN PRINT
 CHR$(219);:X=X-1:IF X<0 THEN X=7
130 IF J$=CHR$(31) THEN PRINT CHR$(219)
;:X=X+1:IF X>7 THEN X=0
140 IF J$=" " THEN 160 ELSE IF J$="F" T
HEN 30
150 GOTO 110
160 LOCATE 0,20:PRINT"                ":LOC
ATE 0,20:INPUT L$:IF LEN(L$)<>8 THEN 16
0 ELSE VPOKE I+X,VAL("&B"+L$):LOCATE 1,
X+3:FOR L=1 TO 8:IF MID$(L$,L,1)="1" TH
EN PRINT CHR$(219); ELSE PRINT CHR$(1);
"G";
170 NEXT L
180 GOTO 110
```

Compare o programa acima com o da figura 4.19 e verifique o que há de diferente entre os dois.

fig. 4.19 - Editor de sprite 16x16 .

```
10 REM Editor de sprites
20 SCREEN 1,3:COLOR 15,4,4:WIDTH 28
30 KEYOFF:CLS:PRINT "Redefinidor de Spr
ites 16x16"
40 PRINT STRING$(18,219):FOR I=1 TO 16:
PRINT CHR$(219);:FOR J=1 TO 16:PRINT CH
R$(1);"G";:NEXT J:PRINT CHR$(219):NEXT
I:PRINT STRING$(18,219)
50 PRINT:PRINT:INPUT "Numero do sprite
";N
60 I=32*N+14336
70 LOCATE 0,3:FOR J=I TO I+15:B$=BIN$(2
56*VPEEK(J)+VPEEK(J+16)):B$=STRING$(16-
LEN(B$),48)+B$:PRINT CHR$(219);
80 FOR K=1 TO 16:IF MID$(B$,K,1)="1" TH
EN PRINT CHR$(219); ELSE PRINT CHR$(1);
"G";
90 NEXT K:PRINT:NEXT J:X=0:PUTSPRITE 0,
(180,40),15,N
100 LOCATE 0,X+3:PRINT ">";CHR$(8);
110 J$=INKEY$:IF J$=CHR$(30) THEN PRINT
 CHR$(219);:X=X-1:IF X<0 THEN X=15
120 IF J$=CHR$(31) THEN PRINT CHR$(219)
;:X=X+1:IF X>15 THEN X=0
130 IF J$=" " THEN 150 ELSE IF J$="F" T
HEN 30
140 GOTO 100
150 LOCATE 0,21:PRINT STRING$(20,32):LO
CATE 0,21:INPUT L$:IF LEN(L$)<>16 THEN
150 ELSE VPOKE I+X,VAL("&B"+LEFT$(L$,8)
):VPOKE I+X+16,VAL("&B"+RIGHT$(L$,8)):L
OCATE 1,X+3:FOR L=1 TO 16:IF MID$(L$,L,
1)="1" THEN PRINT CHR$(219); ELSE PRINT
 CHR$(1);"G";
160 NEXT L
170 GOTO 100
```

Sprites ampliados

Existe também a opção de escrever na tela os sprites com o dobro do tamanho físico. ou seja. a altura e a largura são o dobro. mas o número de pontos é igual ao modo original. Isto é o que chamamos alterar a magnitude do sprite. e é lógico que para que isto seja possível. cada ponto do sprite torna-se duas vezes mais largo e alto (formado por quatro "pixels").

Selecionando modos de sprites

Você deve estar se perguntando como o computador sabe quando deve reconhecer um sprite 8x8 ou 16x16. ou ainda se a magnitude deve ser normal ou ampliada (dobro do tamanho dos pontos).

Uma forma de fazer isso é através do comando:

SCREEN m,n

onde n indica o modo do sprite. Se imaginarmos n como um número binário de dois bits, teremos:

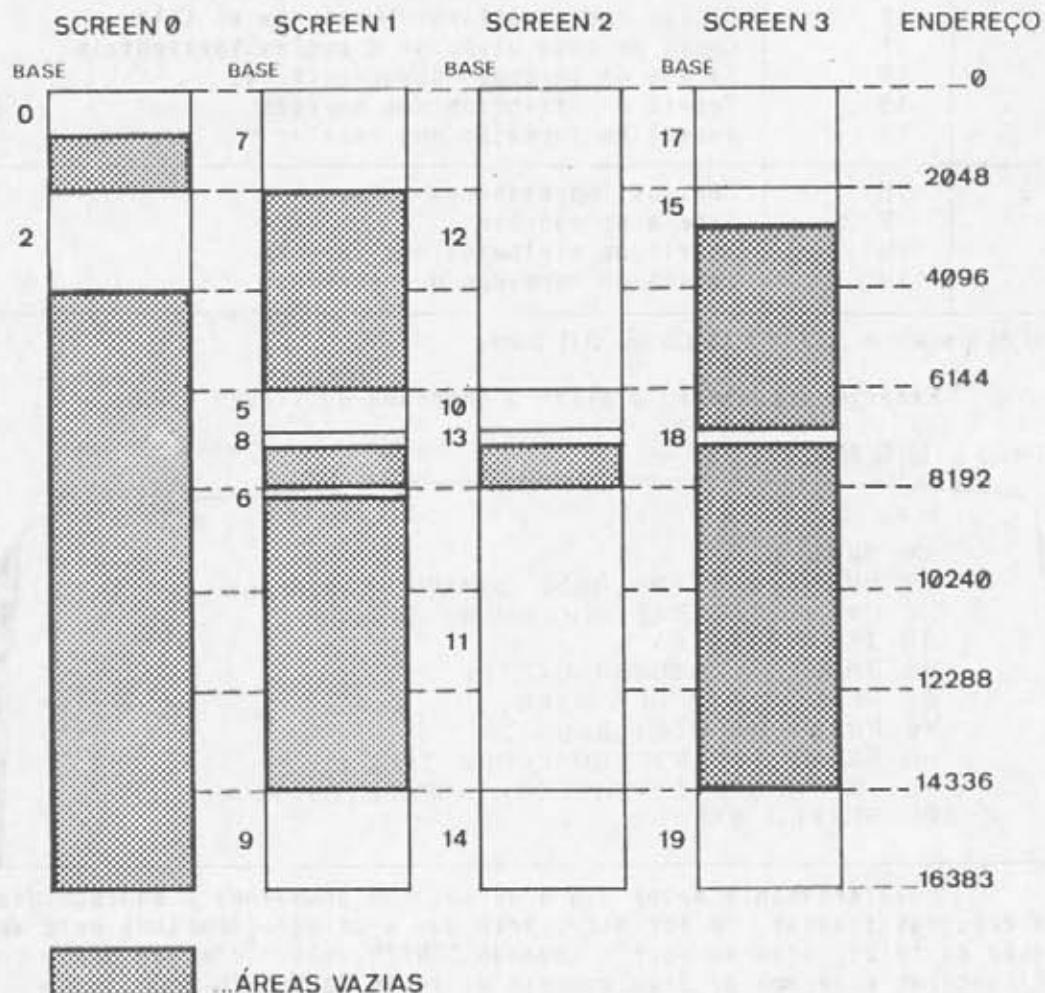
```
n = B1  B0
    |   └──── indica o tamanho lógico (0=8x8. 1=16x16).
    └──────── indica o tamanho físico (0=normal, 1=dobrado).
```

O valor de m pode estar entre 1 e 3 e é opcional (a não ser que você esteja no modo SCREEN 0).

Existe outro modo de definir o tamanho e a magnitude do sprite, com o comando VDP (m) = n. Você verá como funciona este comando nos itens "O comando VDP" e "Os registros do VDP", expostos mais adiante.

Coincidência de sprites.

Um recurso muito interessante em jogos é a possibilidade de se poder detectar a coincidência (ou "colisão") de sprites. Em BASIC, isto é feito com os comandos SPRITE ON e ON SPRITE GOSUB. Mesmo assim, existe um outro modo de pesquisar a coincidência por um método mais interessante, especialmente se você pretende usar programas em Assembly: ler diretamente do VDP a "flag" de coincidência através da leitura do registro de status do VDP (VDP(8)), como mostra o programa da figura 4.20 .

fig. 4.20 - Teste de coincidência de sprites.

```
10 'SIMULACAO DA INSTR. ON SPRITE GOSUB
20 COLOR15,1,1:DEFINTA-F:SCREEN2
30 'DEFINE OS SPRITES,AS CORES E OS NOM
ES DOS SPRITES
40 FOR F=0TO7:READ A:VPOKE(14336+F),A:V
POKE(14344+F),A:NEXT F
50 VPOKE 6915,8:VPOKE 6919,7:VPOKE6914,
0:VPOKE6918,1
60 'DEFINE A FORMA DO MOVIMENTO,POSICAO
    E VERIFICA A FLAG DE SOBREPOSICAO
70 FOR F=0TO255
80 Y1=96+90*SIN(F/64*3.14):Y2=96+90*COS
(F/64*3.14)
90 VPOKE 6912,Y1:VPOKE 6913,F
100 VPOKE 6916,Y2:VPOKE 6917,F
110 PSET(F,Y1),8:PSET(F,Y2),7
120 A=VDP(8):B=A AND 32:IFB=32THEN BEEP
:A=0:B=0
130 NEXT F
140 CLS:GOTO 70
150 RETURN
160 DATA 0,64,124,127,124,64,0,0
```

Para fazer o mesmo em linguagem de máquina, não é possível usar o VDP (8). Entretanto, há outra solução: pesquisar o byte 62439, que contém uma cópia do registro de status do VDP. Se o bit 5 dessa posição valer 1, houve coincidência. Caso contrário, não houve.

O comando BASE

Devido às diferentes estruturas da VRAM para as várias SCREENs, existe a função BASE que permite determinar e/ou redefinir a localização das várias regiões (tabelas) na VRAM para cada modo (fig. 4.21).

fig. 4.21 - Tabelas na VRAM.

A função BASE define, então, o endereço inicial (na VRAM) de cada tabela. Lendo o valor de:

BASE (argumento)

Podemos localizar a posição de cada tabela em cada modo SCREEN. Como a função BASE tem um funcionamento semelhante ao de uma variável, ela pode ser lida e modificada.

A seguir, na figura 4.22, temos uma tabela com os argumentos do BASE e seu significado.

fig. 4.22 - Significado dos vários argumentos do BASE.

SCREEN	ARGUMENTO	DESCRIÇÃO
0	0	Códigos dos caracteres impressos na tela
	2	Tabela de formação dos caracteres
1	5	Códigos dos caracteres impressos na tela
	6	Cores dos "octetos" de caracteres
	7	Tabela de formação dos caracteres
	8	Tabela de atributos dos sprites
	9	Tabela de formação dos sprites
2	10	Código dos caracteres impressos na tela
	11	Cores de cada grupo de 8 pontos horizontais
	12	Tabela de formação dos caracteres
	13	Tabela de atributos dos sprites
	14	Tabela de formação dos sprites
3	15	Códigos impressos na tela
	17	Tabela de padrões
	18	Tabela de atributos dos sprites
	19	Tabela de formação dos sprites

Nota: os argumentos 1, 3, 4 e 16 não são utilizados.

Experimente agora, digitar o programa da figura 4.23 .

fig. 4.23 - Função BASE.

```
 10 SCREEN 0
 20 PRINT BASE(0),BASE(2):VPOKE 150,65
 30 PRINT "APERTE QUALQUER TECLA"
 40 I$=INPUT$(1)
 50 BASE(0)=2048:BASE(2)=0
 60 SCREEN 0:VPOKE 2198,65
 70 PRINT BASE(0),BASE(2)
 80 PRINT "APERTE QUALQUER TECLA"
 90 I$=INPUT$(1):BASE(0)=0:BASE(2)=2048
100 SCREEN 0:END
```

É interessante notar que o cursor não acompanha a mudança das posições das tabelas. Se for necessário que o cursor acompanhe esta variação de telas, deve-se usar o comando SCREEN após o comando BASE, como foi feito no programa de arquivamento de telas da SCREEN 0 .

Experimente usar o BASE com as outras SCREENs.

O comando VDP

Finalmente, temos o comando VDP, cuja função é simplesmente ler ou escrever dados nos registros do chip do VDP, responsáveis por definir as características da tela (cor, modo da SCREEN, etc). Observe que os registros (0 a 7) do VDP só podem ser escritos, enquanto que o registro de STATUS pode somente ser lido. Deste modo, quando dizemos "ler" os registros de 0 a 7, na verdade estamos lendo valores na RAM (variáveis do sistema) que sempre contêm uma cópia de seus valores. Note também que

estes valores da RAM são alterados pelos comandos VDP, COLOR e SCREEN, e não serão alterados se você modificar os registros do VDP em linguagem de máquina, a não ser que você comande isto.

fig. 4.24 - Registros do VDP.

REGISTRO	BITS							
0	0	0	0	0	0	0	M3	EV
1	X	BK	IE	M1	M2	0	SZ	MG
2	0	0	0	0	B3	B2	B1	B0
3	C7	C6	C5	C4	C3	C2	C1	C0
4	0	0	0	0	0	F2	F1	F0
5	0	A6	A5	A4	A3	A2	A1	A0
6	0	0	0	0	0	S2	S1	S0
7	T3	T2	T1	T0	P3	P2	P1	P0
STATUS	F	5S	C	Q4	Q3	Q2	Q1	Q0

Os registros de 0 a 7 podem ser lidos (indiretamente na VRAM) ou escritos diretamente. Já o registro de STATUS pode apenas ser lido. Segue a explicação dos vários bits dos registros.

M1 a M3 definem o modo de operação do VDP:

M1	M2	M3	MODO
0	0	0	SCREEN 1
0	0	1	SCREEN 2
0	1	0	SCREEN 3
1	0	0	SCREEN 0

EV controla um eventual VDP adicional.

X não influi na operação do VDP.

BK quando 0, envia para a TV somente a borda; quando 1, envia toda a imagem.

IE quando 0, desabilita interrupções; quando 1, habilita-as.

SZ quando 0, ativa sprite em 8 x 8 ; quando 1, ativa em 16x16 .

MG quando 0, o sprite é apresentado em tamanho normal; quando 1, o faz em tamanho dobrado.

B0 a B3 definem o endereço base da tabela de códigos dos caracteres. Eles formam os quatro bits mais significativos de uma palavra de 14 bits que é o endereço real da tabela.

C0 a C7 definem o endereço base da tabela de cores. Eles formam os oito bits mais significativos da palavra de 14 bits.

F0 a F2 definem o endereço base da tabela de formação dos caracteres. Formam os três bits mais significativos de 14 bits.

A0 a A6 definem o endereço base da tabela de atributos dos sprites e formam os sete bits mais significativos de 14 bits.

S0 a S2 definem o endereço base da tabela de formação dos sprites. Formam os três bits mais significativos de 14 bits.

T0 a T3 definem o código da cor dos caracteres no modo screen 0.

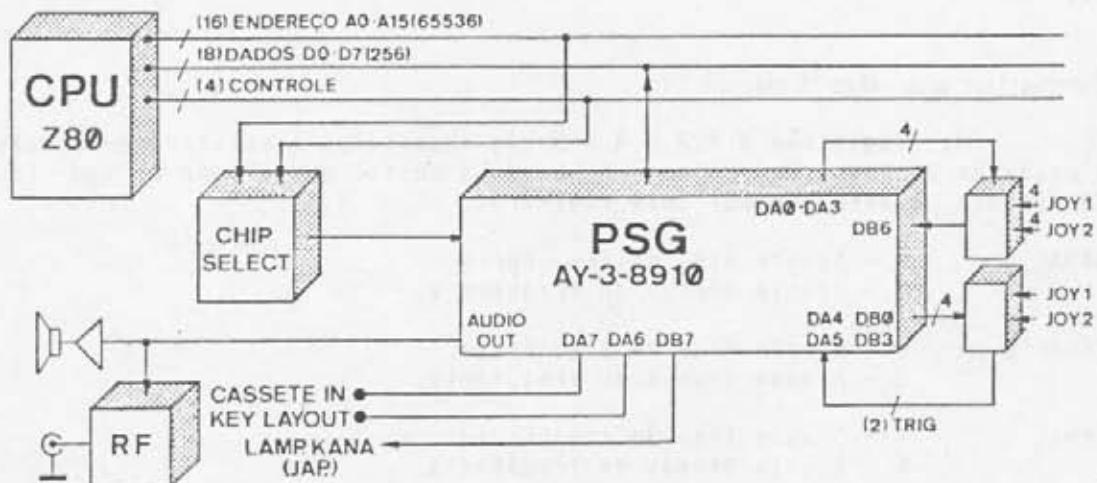
P0 a P3 definem os códigos da cor de fundo (SCREEN 0) e borda nos demais modos.

F é uma flag que indica interrupção. Setado no fim de cada rastreamento.

5S quando 1, indica que há 5 ou mais sprites na mesma horizontal.

C quando 1, indica se dois ou mais sprites são coincidentes.

Q0 a Q4 indicam o número do quinto sprite coincidente, quando 5S vale 1.

Analogamente ao BASE, o VDP pode ser lido ou escrito, indicando o número do registro desejado entre parênteses. Experimente, por exemplo ler os valores dos registros 0, 1 e 7 para verificar o modo da SCREEN e as cores. Vale a pena ressaltar que os comandos do BASIC MSX (COLOR, SCREEN, etc) são por si só bastante rápidos e poderosos, tornando rara a utilização do comando VDP.

Capítulo 5
PSG-GERADOR DE SONS PROGRAMÁVEL

Introdução

No MSX, o circuito responsável por toda a síntese de sons é o PSG (Programmable Sound Generator), ou gerador de sons programável. O coração deste sistema é um circuito integrado AY-3-8910 da General Instruments, sintetizador dotado de três canais (A, B e C) que podem ser controlados independentemente em volume, frequência, tipo de som e envelope.

A estrutura de hardware do PSG está mostrada na figura 5.1.

fig. 5.1 - Estrutura esquemática do hardware do PSG.

A grande versatilidade deste circuito é notória pois, além de gerar sons, ele também controla a leitura do estado dos joysticks e a entrada dos sinais do cassete durante a leitura de dados da fita.

Funcionamento e operação

O controle do PSG é feito através de 14 registros (0 a 13), sendo que cada um deles tem uma função bem específica. O valor desses registros não pode ser lido, mas pode ser alterado pelo comando SOUND. Por exemplo, quando o usuário digita:

```
SOUND 7,56
```

o registro 7 recebe o valor 56 (mais adiante veremos as funções de cada registro).

Um outro modo de programar os registros é através do comando OUT (do BASIC ou do Assembly). Para isso é necessário conhecer como a CPU ativa o PSG.

O PSG é mapeado como um periférico, portanto o controle deve ser efetuado por uma ou mais portas de I/O da CPU. De fato, duas portas foram resevadas para este fim: a porta 160 (&HA0), que seleciona o registro a ser alterado, e a porta 161 (&HA1), que transmite o dado a ser escrito. Assim, o comando mostrado acima poderia ser substituído por:

```
OUT 160,7
OUT 161,56
```

em BASIC, ou

```
LD A,7          3E 07
OUT (160),A     D3 A0
LD A,56         3E 38
OUT (161),A     D3 A1
```

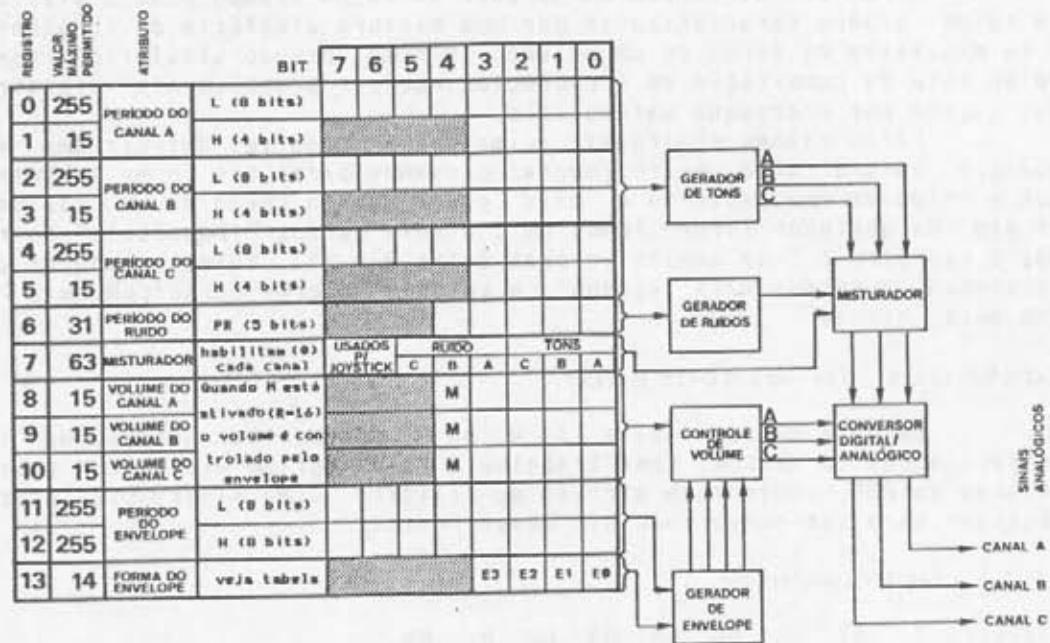
em Assembly.

Os registros do PSG.

Como já vimos, o PSG possui 14 registros de controle (0 a 13). A estrutura destes registros é apresentada na figura 5.2.

Veremos agora como cada um deles funciona e para o que podemos usá-los.

Geradores de tons

Os registros 0,1,2,3,4 e 5 são reservados especificamente para a produção de tons. Cada canal (A, B ou C) possui um gerador de tom independente, controlado por dois registros:

CANAL A
- 0 - Ajuste fino de freqüência.
- 1 - Ajuste grosso de freqüência.

CANAL B
- 2 - Ajuste fino de freqüência.
- 3 - Ajuste grosso de freqüência.

CANAL C
- 4 - Ajuste fino de freqüência.
- 5 - Ajuste grosso de freqüência.

O valor máximo que pode ser atribuído aos registros pares (ajuste fino) é 255 e, aos ímpares, é 15 . Assim, a frequência mais baixa que pode ser gerada pelo oscilador de áudio do PSG é mais ou menos 30 Hz e a máxima pode ultrapassar os 100 kHz!

fig. 5.2 - Registros do PSG.

REGISTRO	VALOR MÁXIMO PERMITIDO	ATRIBUTO	BIT	7	6	5	4	3	2	1	0
0	255	PERÍODO DO CANAL A	L (8 bits)								
1	15		H (4 bits)								
2	255	PERÍODO DO CANAL B	L (8 bits)								
3	15		H (4 bits)								
4	255	PERÍODO DO CANAL C	L (8 bits)								
5	15		H (4 bits)								
6	31	PERÍODO DO RUÍDO	PR (5 bits)								
7	63	MISTURADOR	habilitam (0) cada canal	USADOS P/ JOYSTICK		RUÍDO C	RUÍDO B	RUÍDO A	TONS C	TONS B	TONS A
8	15	VOLUME DO CANAL A	Quando M está ativado (M=16) o volume é controlado pelo envelope				M				
9	15	VOLUME DO CANAL B					M				
10	15	VOLUME DO CANAL C					M				
11	255	PERÍODO DO ENVELOPE	L (8 bits)								
12	255		H (8 bits)								
13	14	FORMA DO ENVELOPE	veja tabela					E3	E2	E1	E0

GERADOR DE TONS (A, B, C) — GERADOR DE RUÍDOS — MISTURADOR — CONTROLE DE VOLUME (A, B, C) — CONVERSOR DIGITAL/ANALÓGICO — SINAIS ANALÓGICOS — CANAL A — CANAL B — CANAL C — GERADOR DE ENVELOPE

A freqüência do tom de um canal em função dos seus registros pode ser determinada pela fórmula:

Ft=Fc/(32*(Rf+256*Rg))

onde: Ft= Freqüência do tom
Fc= Freqüência de Clock da CPU
(Expert = 3 575 611 Hz)
(Hot-Bit = 3 579 545 Hz)
Rf= Registro de ajuste fino (par)
Rg= Registro de ajuste grosso (ímpar)

Reciprocamente, podemos obter os valores dos registros de um canal a partir de uma freqüência desejada:

```
Tt=32*Ft/Fc
Rf=Tt MOD 256
Rg=Tt\256
```

onde: Tt= Período do tom.

Por exemplo, se quisermos que o PSG toque a nota LÁ (440 Hz), teremos que fazer:

```
Rf=254
Rg=0
```

pois:
Fc/(32*(254+256*0))= 439.91 (no Expert) e 440.39 (no Hot-Bit)

isto é, obtemos um valor muito próximo de 440 Hz. A menos que você seja o Isaac Karabtchevsky, a diferença será imperceptível (0,02%)!

Gerador de ruído

O PSG possui também um gerador de ruído branco pseudo-aleatório (o ruído branco caracteriza-se por uma mistura aleatória de freqüências e se manifesta na forma de um chiado). O nome "pseudo-aleatório" é devido ao fato da combinação de freqüências não ser propriamente aleatória, mas gerada por processos matemáticos.

Pelas razões mostradas acima não é possível definir uma freqüência para o ruído, muito embora possamos dizer (de forma subjetiva) que o ruído de uma explosão é mais "grave" que o ruído de um vazamento de gás. De qualquer forma, temos um controle para a "freqüência" do ruído: o registro 6 (que admite valores entre 0 e 31). Valores menores correspondem a ruídos mais "agudos", e valores maiores correspondem a ruídos mais "graves".

Controle do misturador

Em cada um dos canais (A, B ou C) podemos ter geração de tom, ruído, ambos ou nenhum. Esse trabalho é feito por um misturador (mixer) interno do PSG, controlado através do registro 7. Na figura 5.3 pode-se observar para que serve cada bit desse registro.

fig. 5.3 - Registro misturador.

```
Registro 7:  B7  B6  B5  B4  B3  B2  B1  B0
             |   |   |   |   |   |   |   +--- Seleciona tom no canal A
             |   |   |   |   |   |   +------- Seleciona tom no canal B
             |   |   |   |   |   +----------- Seleciona tom no canal C
             |   |   |   |   +--------------- Seleciona ruído no canal A
             |   |   |   +------------------- Seleciona ruído no canal B
             |   |   +----------------------- Seleciona ruído no canal C
             |   +--------------------------- Usado pelos joysticks
             +------------------------------- Usado pelos joysticks
```

Os bits de B0 a B5 possuem lógica negativa, ou seja, quando estão em nível 0 a função correspondente está ativada; quando em 1, a função está desativada.

Os bits B6 e B7 fazem a leitura dos joysticks e não serão tema de estudo neste livro (mesmo porque a CPU faz "mágicas" para ler algo de útil neles e existem maneiras mais inteligentes de se ler joysticks do que pesquisar seus valores diretamente).

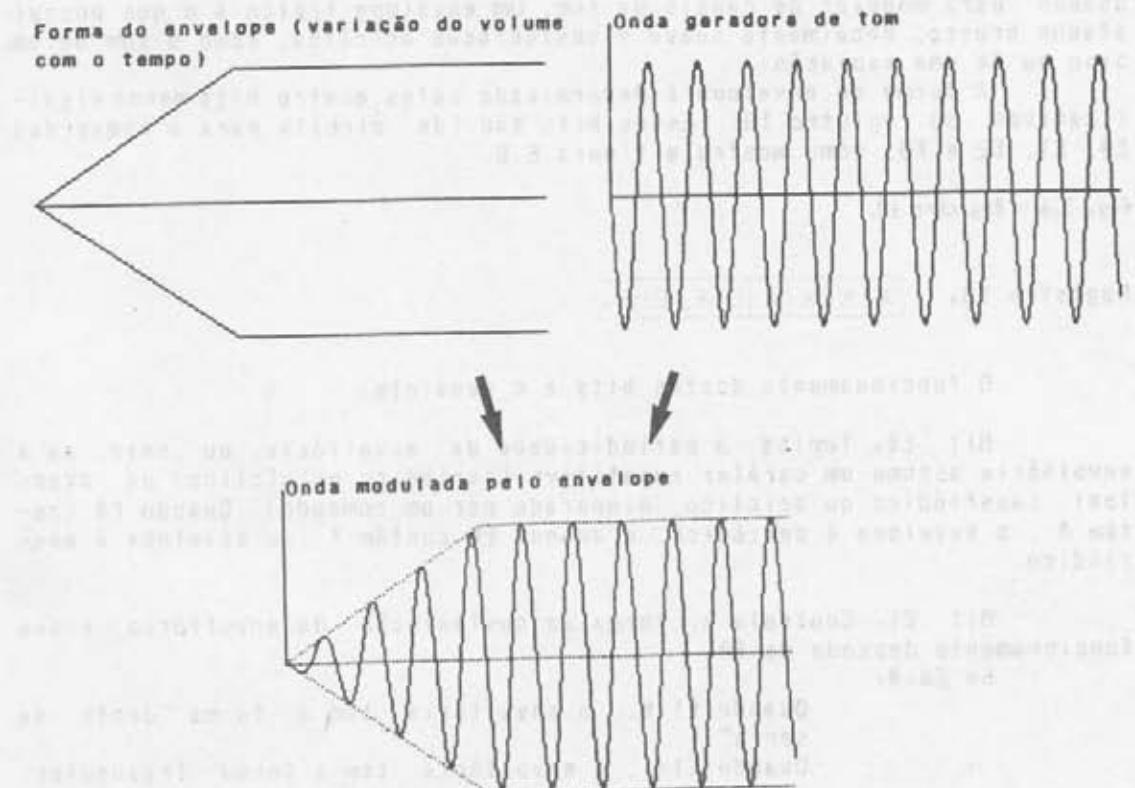
Se, por exemplo, você quiser usar tom no canal A, tom e ruído no canal B e só ruído no canal C, a configuração do registro 7 deve ser semelhante a mostrada na figura 5.4 .

Para programar o registro 7 com a configuração da figura 5.4 pode-se usar um dos quatro números: 12, 76, 140 ou 204 . Podemos, por exemplo, fazer:

```
SOUND 7 , 12
```

Para permitir uma visualização imediata do conteúdo e da função de cada bit do registro 7 , podemos usar o comando SOUND com números binários:

```
SOUND 7 , &B00001100
```

fig. 5.4 - Exemplo de configuração do registro 7.

Controle de volume

O volume de cada canal é determinado pelos registros 8 (canal A), 9 (canal B) e 10 (canal C). O valor que deve ser enviado a esses registros pode variar entre 0 e 15, sendo que 0 corresponde ao volume mínimo (silêncio) e 15 ao volume máximo. Para esta faixa de valores o sinal enviado não sofre modulação, ou seja, é totalmente contínuo.

Existe também a possibilidade de se enviar ao misturador o sinal modulado em volume, ou seja com sua intensidade variando automaticamente com o tempo, seguindo um padrão de envoltória (ou "envelope").Veja isso com mais detalhes nos itens referentes ao controle de envoltória.

Para fazer com que o volume de um canal siga o envelope, basta atribuir valores maiores que 15 (de 16 a 255) ao seu registro de volume. Na fig.5.5 pode-se ver como um sinal "senoidal" é modulado por uma envoltória acíclica (isto é, que não se repete).

fig. 5.5 - Sinal modulado.

Inicialmente o sinal tem uma intensidade bastante baixa que vai aumentando até atingir um patamar do qual não sai mais. Esse envelope é semelhante ao do som produzido por um órgão do tubos.

Período do envelope

O período do envelope é controlado pelos registros 11 e 12 de modo análogo ao controle de frequência de tons.

Para obter uma frequência a partir dos registros 11 e 12, usa-se a fórmula:

Fe=Fc/512/(R11+256*R12)

onde:

Fe = Frequência do envelope
R11 = Conteúdo do registro 11
R12 = Conteúdo do registro 12
Fc = Frequência do clock da CPU (veja página 111)

Reciprocamente, dada uma frequência, podemos determinar os valores de R11 e R12 que a definem.

Vr=Fc/512/Fe
R11=Vr MOD 256
R12=VR 256

O registro 11 é o ajuste fino e o registro 12 é o ajuste grosso.

Formato do envelope

Existem oito formatos de envoltória (ou envelope) que podem ser usados para modular os canais de som. Um envelope típico é o que possui ataque brusco, decaimento suave e sustentação acíclica, como o som de um sino ou de uma explosão.

A forma do envelope é determinada pelos quatro bits menos significativos do registro 13. Esses bits são (da direita para a esquerda) E0, E1, E2 e E3, como mostra a figura 5.6.

fig. 5.6 - Registro 13.

Registro 13:

X	X	X	X	E_3	E_2	E_1	E_0

O funcionamento destes bits é o seguinte:

Bit E0: Indica a periodicidade da envoltória, ou seja, se a envoltória assume um caráter repetitivo (periódico ou cíclico) ou eventual (aperiódico ou acíclico, disparado por um comando). Quando E0 contém 0 , o envelope é periódico, e quando E0 contém 1 , o envelope é aperiódico.

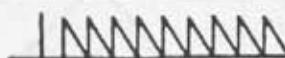
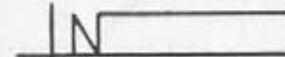
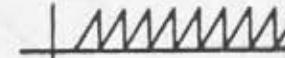
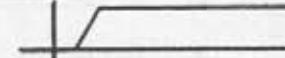
Bit E1: Controla a forma da sustentação da envoltória, e seu funcionamento depende de E0:

Se E0=0:

Quando E1=0, a envoltória tem a forma "dente de serra".
Quando E1=1, a envoltória tem a forma triangular.

Se E0=1:

Quando E1=0, a sustentação acompanha o ataque.
Quando E1=1, a sustentação não acompanha o ataque.

Bit E2: Indica como se processa o ataque. Se E2 contém 0, o ataque inicia com volume mínimo e termina com volume máximo. Se E2 contém 1, o ataque inicia com volume máximo e vai até o mínimo.

Bit E3: Controla as prioridades dos três bits anteriores. Quando E3 contém 0, vale apenas a prioridade do bit E2 e, neste caso:

Se E2=0, a envoltória é acíclica com ataque decrescente e com a sustentação seguindo o ataque.

Se E2=1, a envoltória é acíclica com ataque crescente e sustentação descontínua ao ataque.

Na figura 5.7 pode-se observar as várias formas de envoltória.

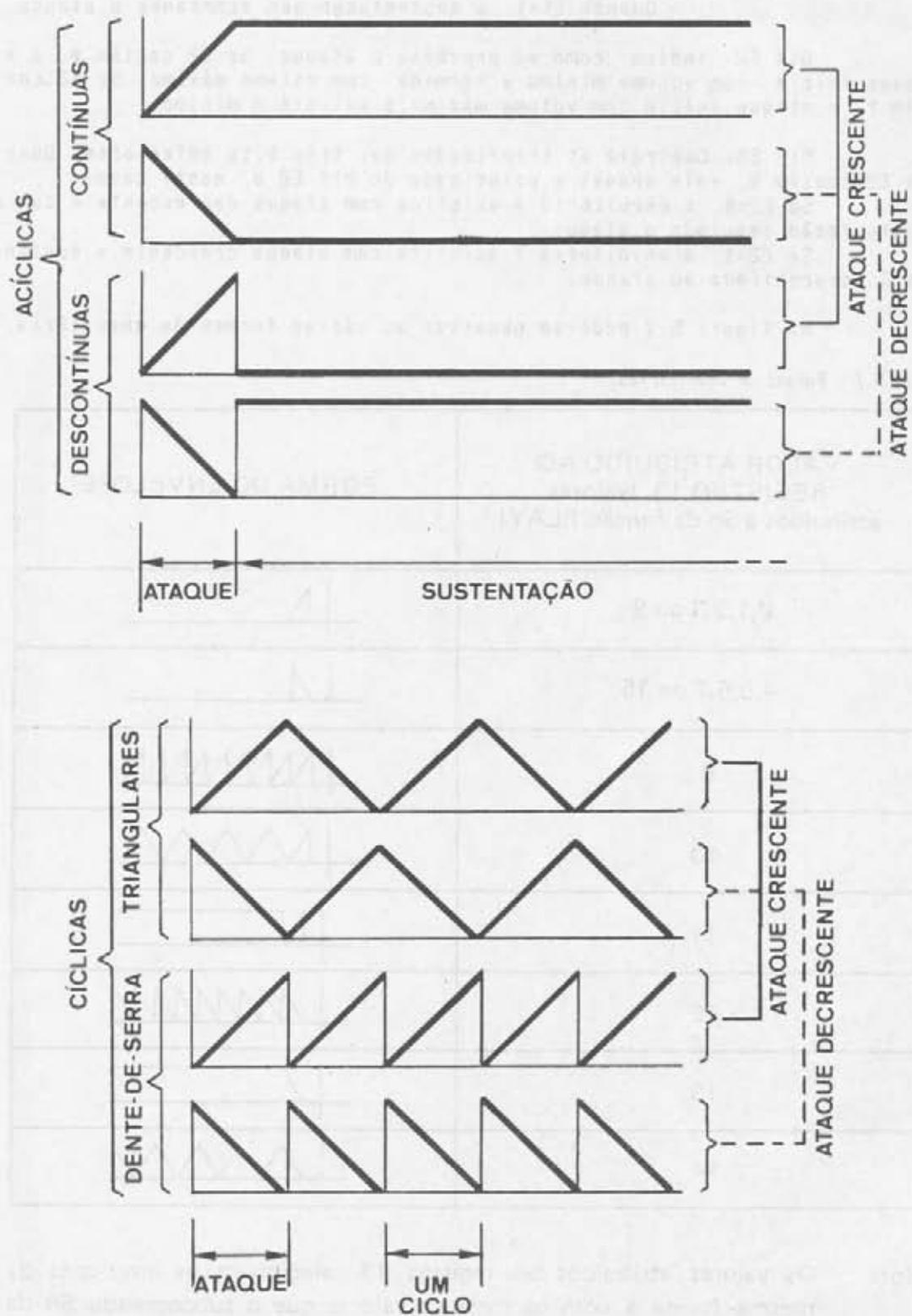



fig. 5.7 - Formas de envoltórias.

VALOR ATRIBUÍDO AO REGISTRO 13 (valores atribuídos a Sn da função PLAY)	FORMA DO ENVELOPE
Ø,1,2,3 ou 9	
4,5,6,7 ou 15	
8	
10	
11	
12	
13	
14	

Nota: Os valores atribuídos ao registro 13 selecionam os envelopes da mesma forma e com os mesmos valores que o subcomando Sn da função PLAY.

fig. 5.8 - Classificação dos envelopes.

Programando o PSG

Após estas explicações, vamos apresentar um programa que manipula todos os registros: o editor de sons. Este programa tem como função dar ao usuário um controle simples (porém poderoso) dos diversos registros do PSG, eliminando a tarefa de usar o comando SOUND, por vezes de difícil acesso.

O editor de sons nada mais faz que alterar os valores dos registros correspondentes aos campos que aparecem na tela, permitindo um fácil controle visual.

fig. 5.9 - Editor de sons.

```
10 REM Editor de Sons
20 RA=0:RB=0:RC=0:SA=0:SB=0:SC=0:VA=0:V
B=0:VC=0:E1=0:E2=0:F0=0:A1=0:A2=0:B1=0:
B2=0:C1=0:C2=0:FA=0:FB=0:FC=0:FR=0:S$="
T":FOR I=1 TO 10:KEY(I) ON:NEXT I:ON KE
Y GOSUB 90,150,200,230,240,260,290,310,
320,330
30 M=1:FOR I=1 TO 9:READ XC(I),YC(I):NE
XT I
```

```
40 CLS:KEYOFF:SCREEN 1:WIDTH 28:PRINT T
AB(8);"EDITOR DE SONS":LOCATE 11,3:PRIN
T "C A N A L":PRINT:PRINT TAB(8);"A
  B      C":PRINT:PRINT ">Tom":PRINT:PR
INT " Ruido":PRINT:PRINT " Freq":PRINT:
PRINT " R+":PRINT:PRINT " R-":PRINT:PRI
NT " Volume"
50 PRINT:PRINT " Envelope T-        T+":P
RINT:PRINT " Forma          Ruido":DEF FN
A(X)=111875!/X\256:DEF FNB(X)=INT(.5+11
1875!/X MOD 256)
60 LOCATE 7,7:PRINT A1;TAB(14);B1;TAB(2
1);C1:LOCATE 7,9:PRINT A2;TAB(14);B2;TA
B(21);C2:LOCATE 7,11:PRINT FA;TAB(14);F
B;TAB(21);FC:LOCATE 7,13:PRINT RA;TAB(1
4);RB;TAB(21);RC:LOCATE 7,15:PRINT SA;T
AB(14);SB;TAB(21);SC
70 LOCATE 7,17:PRINT VA;TAB(14);VB;TAB(
21);VC:LOCATE 12,19:PRINT E1:LOCATE 20,
19:PRINT E2:LOCATE 7,21:PRINT F0:LOCATE
 19,21:PRINT FR
80 GOTO 80
```

```
90 IF M=1 THEN A1=1-A1 ELSE IF M=2 THEN
 A2=1-A2 ELSE IF M=3 THEN 140 ELSE IF M
=4 THEN RA=RA-1 ELSE IF M=5 THEN SA=SA-
1 ELSE IF M=6 THEN VA=VA-1 ELSE IF M=8
THEN F0=F0-1
100 IF M=9 THEN FR=FR-1
110 IF M=7 THEN E1=E1-1
120 IF E1<0 THEN E1=0
130 IF M<>3 THEN 340
140 LOCATE 0,22:INPUT "Freq ";FA:RA=FNA
(FA):SA=FNB(FA):GOTO 340
150 IF M=1 THEN B1=1-B1 ELSE IF M=2 THE
N B2=1-B2 ELSE IF M=3 THEN 190 ELSE IF
M=4 THEN RB=RB-1 ELSE IF M=5 THEN SB=SB
-1 ELSE IF M=6 THEN VB=VB-1
160 IF M=7 THEN E2=E2-1
170 IF E2<0 THEN E2=0
180 IF M<>3 THEN 340
190 LOCATE 0,22:INPUT "Freq ";FB:RB=FNA
(FB):SB=FNB(FB):GOTO 340
200 IF M=1 THEN C1=1-C1 ELSE IF M=2 THE
N C2=1-C2 ELSE IF M=3 THEN 220 ELSE IF
M=4 THEN RC=RC-1 ELSE IF M=5 THEN SC=SC
-1 ELSE IF M=6 THEN VC=VC-1
210 IF M<>3 THEN 340
220 LOCATE 0,22:INPUT "Freq ";FC:RC=FNA
(FC):SC=FNB(FC):GOTO 340
230 SOUND 13,F0:RETURN 60
240 LOCATE XC(M),YC(M):PRINT " ":M=M+1:
IF M>9 THEN M=1
250 LOCATE XC(M),YC(M):PRINT ">":RETURN
 60
260 IF M=4 THEN RA=RA+1 ELSE IF M=5 THE
N SA=SA+1 ELSE IF M=6 THEN VA=VA+1 ELSE
 IF M=8 THEN F0=F0+1 ELSE IF M=7 THEN E
1=E1+1
270 IF M=9 THEN FR=FR+1
280 GOTO 340
290 IF M=4 THEN RB=RB+1 ELSE IF M=5 THE
N SB=SB+1 ELSE IF M=6 THEN VB=VB+1 ELSE
 IF M=7 THEN E2=E2+1
300 GOTO 340
310 IF M=4 THEN RC=RC+1 ELSE IF M=5 THE
N SC=SC+1 ELSE IF M=6 THEN VC=VC+1
320 GOTO 340
330 RETURN 60
340 LOCATE 0,22:PRINT "
":IF VA<0 THEN VA=0 ELSE IF VA>16 THEN
VA=16
```

```
350 IF VB<0 THEN VB=0 ELSE IF VB>16 THE
N VB=16
360 IF VC<0 THEN VC=0 ELSE IF VC>16 THE
N VC=16
370 IF RA<0 THEN RA=0 ELSE IF RA>15 THE
N RA=15
380 IF RB<0 THEN RB=0 ELSE IF RB>15 THE
N RB=15
390 IF RC<0 THEN RC=0 ELSE IF RC>15 THE
N RC=15
```

```
400 IF SA<0 THEN SA=0 ELSE IF SA>255 TH
EN SA=255
410 IF SB<0 THEN SB=0 ELSE IF SB>255 TH
EN SB=255
420 IF SC<0 THEN SC=0 ELSE IF SC>255 TH
EN SC=255
430 IF FR<0 THEN FR=0 ELSE IF FR>31 THE
N FR=31
440 IF E1<0 THEN E1=0 ELSE IF E1>255 TH
EN E1=255
450 IF E2<0 THEN E2=0 ELSE IF E2>255 TH
EN E2=255
460 S=255-8*A2-16*B2-32*C2-A1-2*B1-4*C1
:SOUND 0,SA:SOUND 1,RA:SOUND 2,SB:SOUND
 3,RB:SOUND 4,SC:SOUND 5,RC
470 IF F0=15 THEN F0=0 ELSE IF F0=-1 TH
EN F0=14
480 SOUND 6,FR:SOUND 7,S:SOUND 8,VA:SOU
ND 9,VB:SOUND 10,VC:SOUND 11,E1:SOUND 1
2,E2
490 FOR I=1 TO 10:KEY(I) ON:NEXT I:RETU
RN 60
500 DATA 0,7,0,9,0,11,0,13,0,15,0,17,0,
19,0,21,13,21
```

Após digitar o programa, comande RUN. Aparecerá na tela um menu com as funções de cada registro (e seus valores correntes) e uma seta apontando para a palavra "Tom". Nesse instante, todos os canais estão com os tons desativados, o que pode ser visto pela seqüência de três zeros da linha. Para ativar o tom no canal A, pressione a tecla F1. O número correspondente ao tom do canal A deverá mudar para 1, deixando-o ativo. Um novo toque em F1 fará este valor voltar a zero, e assim por diante. O mesmo pode ser feito com os canais B e C, bastando usar as teclas F2 e F3 de maneira análoga. Vamos agora gerar alguns tons; faça com que os três canais fiquem ativos (em nível 1). Não se preocupe se não surgir nenhum som por enquanto, pois isto é normal.

Pressione agora a tecla F5. Você verá que a seta não está mais apontando para a palavra "Tom", mas sim para a palavra "Ruído". O funcionamento deste modo é igual ao anterior, com as teclas F1, F2 e F3, uma para cada canal. Experimente pressioná-las algumas vezes. Agora,

desative os três canais de ruído e pressione novamente F5.

Você agora deve estar no modo freqüência ("Freq"). Para entrar a frequência (em Hz) do canal A, use F1 . Para o canal B, F2 , e para o canal C, use F3. Ao pressionar uma dessas teclas, o computador pedirá o valor da freqüência correspondente. Entre com os seguintes valores: canal A=440, canal B=880 e canal C=1760. Note que os registros de freqüência R+ e R- (registros 1, 3 e 5 e 0, 2 e 4) são alterados conforme a fórmula mostrada anteriormente.

Para ouvir o resultado disto, é preciso aumentar os volumes dos canais (pressione F5 três vezes para entrar no modo "Volume"). Não se esqueça também de aumentar o volume da saída de áudio do micro (no Expert) ou o volume da TV.

Para aumentar o volume do canal A, pressione F6 (=SHIFT+F1), e para diminuir, use somente F1. Para o canal B, valem as teclas F7 e F2, e para o canal C, F8 e F3. Ajuste o volume 10 para os três canais (o valor máximo é 15).

Agora que você está ouvindo este acorde, pressione F5 repetidas vezes até voltar ao modo Freqüência. Experimente mudar as freqüências para os seguintes valores:

A=100 Hz, B=110 Hz, C=120 Hz
ou: A=440 Hz, B=450 Hz, C=460 Hz

Vamos agora manipular os registros de freqüência diretamente. Pressione F5 para mudar para o modo R+ (registros 1, 3 e 5). Neste instante a tecla F6 (SHIFT+F1) aumenta o valor de R1 e F1 diminui; F7 aumenta R3 e F2 diminui; F8 aumenta R5 e F3 diminui. Faça experiências com isto.

Para fazer o ajuste fino, vá para o modo R- (registros 0, 2 e 4) e proceda analogamente ao modo R+.

Bem, isto esgota o controle de tom. Vamos experimentar o controle de ruído e, para isto, desative o tom dos três canais e ative somente o ruído do canal A. Não é necessário ativar o ruído em B e C, pois a freqüência deste é sempre igual nos três canais. Note como o ruído se assemelha ao barulho da chuva.

Você pode também variar o tipo do ruído, passando para o modo "Ruído" e alterando seu valor (F6 aumenta, F1 diminui). Tente usar valores entre 0 e 31 (com valor 31, o ruído se assemelha ao barulho de um avião a jato).

Um recurso interessante do PSG é o gerador de envelope. Vamos fazê-lo funcionar para simular o ruído de uma explosão. Passe o programa para o modo "Volume" e ajuste o volume do canal A para o valor 16 (neste instante, o ruído deverá cessar). Ao passar o volume do canal A para 16, este fica subordinado ao gerador de envelope.

O próximo passo é selecionar o período correspondente ao de uma explosão. Entre no modo T+ (ajuste grosso do período) com auxílio de F5 e, com auxílio de F6, ajuste seu valor para 64.

Outro ajuste que deveria ser feito é o da forma do envelope (registro 13), mas este não é necessário neste momento pois quando o programa é iniciado (com todos os registros contendo 0), fica definida automaticamente a forma da explosão.

Otimo! Agora só falta disparar. Faça-o pressionando a tecla F4, o que deverá causar o ruído de uma explosão. Experimente mudar o valor da forma do envelope para, por exemplo, 10. Mude também as variáveis T+ e T- (registros 1 e 12), e você notará que o valor de T- (R11) pouco altera o período do envelope. Tente também modular tons pelo envelope. Como você faria, por exemplo, o barulho de um sino? E de uma Maria-fumaça?

Capítulo 6
DISPOSITIVOS DE ARMAZENAMENTO

O gravador cassete

O sistema de armazenamento em fita cassete não é certamente o mais perfeito possível: ele apresenta várias falhas, como a falta de confiabilidade, a demora no tempo de leitura e o fato de ser um arquivo sequencial. Em compensação ele se constitui no sistema mais econômico de arquivo.

Sendo ,porém, um sistema razoavelmente sujeito à falhas, existem algumas medidas que podem ser tomadas para torná-lo mais confiável.

Cuidado com o gravador e as fitas

1) É essencial que você possua um bom gravador. Um dos melhores do mercado é o Datacorder que apesar de ser um bom gravador, apresenta um grave problema: é movido a pilhas.

Quando as pilhas começam a ficar fracas, a velocidade do gravador é seriamente comprometida. Você pode ter um programa gravado corretamente,mas como a rotação do gravador está menor, o micro poderá não reconhecer os sinais gravados.

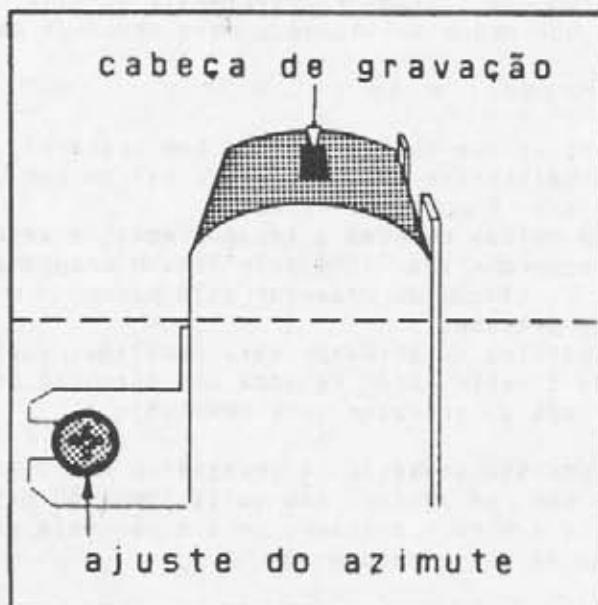
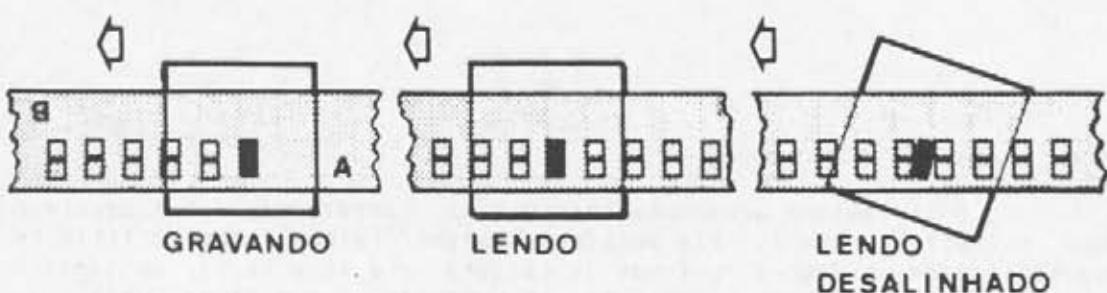
A melhor maneira de eliminar este problema, é adquirir um eliminador de pilhas de 6 volts. Além de você não depender mais das pilhas, a velocidade de rotação do gravador será constante.

2) Para uma boa gravação, é necessário uma fita de boa qualidade, que reproduza bem os agudos, não solte emulsão magnética, o que prejudica o cabecote e o rolo pressor, e que não seja de longa duração (as C-20 ou C-30 são as mais indicadas).

3) A limpeza física e magnética dos cabeçotes (leitura/gravação e apagamento) e do rolo pressor é necessária para uma boa qualidade do som.

Para a limpeza física, semanal, um ótimo material é o álcool isopropílico (que não contém água), mas o álcool comum também apresenta um ótimo desempenho nessa função.

Cuidado!, NÃO introduza fitas com o cabeçote molhado: espere o álcool evaporar completamente.

4) A poluição pode atacar também as suas fitas. Existem dois tipos básicos de poluição, a físico-química e a magnética.

A poluição magnética, embora invisível, é altamente prejudicial para os dados em fita. Evite colocá-las perto demais da televisão, altofalantes, reatores de lâmpadas fluorescentes ou transformadores. O eliminador de pilhas também é um transformador, portanto CUIDADO!!!!

A poluição físico-química ataca a emulsão magnética da sua fita, quando ela está fora do gravador. Partículas de poeira ou fuligem podem aderir à fita pois são facilmente atraídas por campos magnéticos.

Outro poluente sério é o solvente de produtos em spray. Quando eles caem na fita, dissolvem o óxido férrico e fazem com que ele, posteriormente, grude no cabeçote, comprometendo a qualidade do som.

Portanto SEMPRE que for retirar a fita do gravador rebobine-a antes, pois o trecho inicial(plástico) envolve o rolo de fita, protegendo-o.

fig. 6.0 Alinhamento de cabeçote.

5) Quanto à leitura de fitas de outros gravadores, o problema pode ser maior que o simples ajuste de volume. Na grande maioria das vezes, o problema é o alinhamento do cabeçote (ou azimute), que pode ser diferente no momento da gravação e por ocasião da leitura em um outro gravador.

Se os dados forem lidos em outros gravadores com a altura do cabeçote diferente, acontecerá o seguinte problema:

Como a área da fita que contém o dado não passa inteiramente pela cabeça de leitura gravação, apenas certas frequências de um dado serão passadas para o micro, eventualmente misturadas com as do dado posterior.

A única maneira de contornar este problema é ajustando o azimute do gravador (figura 6.0).

Com uma pequena chave de fenda você deve movimentar o parafuso no sentido horário ou anti-horário até obter o som mais estridente possível.

Quando mudar o azimute do seu gravador para conseguir ler um programa lembre-se de voltá-lo ao estado original, porque as fitas gravadas com o ajuste antigo não servirão no atual.

Se você possui um grupo de amigos que trocam programas, é bom que vocês tenham os gravadores ajustados com o mesmo azimute, assim uma fita poderá ser lida por todos os gravadores sem a necessidade do ajuste ou do transporte do gravador.

Armazenamento em fita

Você já sabe como um programa fica "guardado" na RAM depois de ter sido digitado.

Se você se "aprofundou" bastante na RAM com os programas dos capítulos anteriores, deve ter notado o quanto é diferente o programa listado no vídeo daquilo que há na RAM.

Devido a essas diferenças, existem várias maneiras para gravar dados em fita cassete:

Você pode gravar dados em arquivos pelos comandos OPEN (modo output) PRINT# e CLOSE. Os programas em BASIC ou linguagem de máquina podem ser salvos através dos comandos BSAVE, CSAVE e SAVE. A recuperação é feita pelos comandos OPEN (modo input) para arquivos, e BLOAD, CLOAD e LOAD para programas em BASIC ou Linguagem de Máquina.

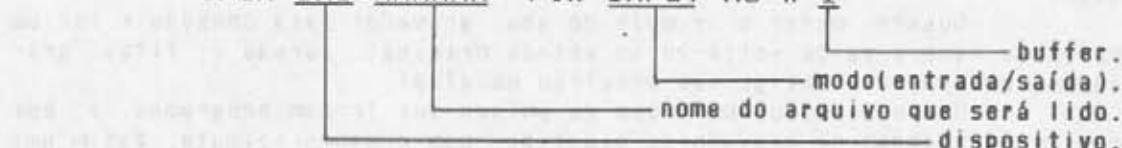
Cada um deles grava as informações de maneira diferente.

Vamos a um breve comentário sobre cada um.

ARQUIVOS EM CASSETE

Para gravar um arquivo em cassete, primeiramente você deve abrir o arquivo pelo comando OPEN, e especificar o modo OUTPUT já que você irá gravá-lo. A sintaxe do comando OPEN está exemplificada na figura 6.1

figura 6.1 Sintaxe do comando OPEN (output)

Quando o micro executa esse comando, uma região de 256 bytes (buffer) é reservada exclusivamente para guardar os dados. O computador também aciona o gravador ou o drive, gravando um cabeçalho que indica o tipo de gravação (no caso um arquivo) e o nome do arquivo.

O comando PRINT# armazena dados no buffer (leia-se bãffer). Quando ele estiver lotado, o micro paralisará todas as operações, acionará o gravador, armazenando na fita os códigos ASCII contidos no buffer, e ao retornar, este é limpo e a execução do programa continua.

Se o buffer não estiver totalmente preenchido, e o comando CLOSE for executado, o micro gravará na fita todo o seu conteúdo.

Para recuperar um arquivo em cassete,basta abrir o arquivo no modo INPUT como mostra a figura 6.2

figura 6.2 Sintaxe do comando OPEN (input).

A partir do momento da execução desse comando, o micro vasculhará a fita até encontrar um arquivo com o nome especificado.

Ao achar o arquivo, os blocos de 256 bytes serão trazidos um a um para o buffer, à medida que forem sendo lidos pelo comandoINPUT#.

Sempre que for trabalhar com arquivos em cassete, certifique-se que o remote do gravador esteja conectado, por que assim o micro lerá automaticamente todos os blocos do arquivo, enquanto que manualmente (ligando e desligando o gravador) você poderá perder algum bloco de um arquivo.

CSAVE

O comando CSAVE"NOME" grava um programa BASIC em fita. Por isso utilizamos CSAVE para gravar um programa depois de tê-lo digitado.

O BASIC MSX mantém informações nas váriáveis do sistema de modo que a todo instante ele "sabe" em quais endereços da RAM começa e termina o seu programa. E' por esse motivo que quando utilizamos CSAVE só precisamos dar o nome do programa, pois o Sistema Operacional já sabe qual o bloco de RAM que deve ser gravado.

Para se ler da fita um programa gravado com CSAVE, deve-se usar o comando CLOAD.

Em resumo, CSAVE "salva" em fita o bloco de memória RAM na qual está seu programa BASIC, copiando byte por byte sem fazer nenhuma alteração. Trata-se de uma gravação mais econômica, pois as palavras reservadas são representadas por tokens.

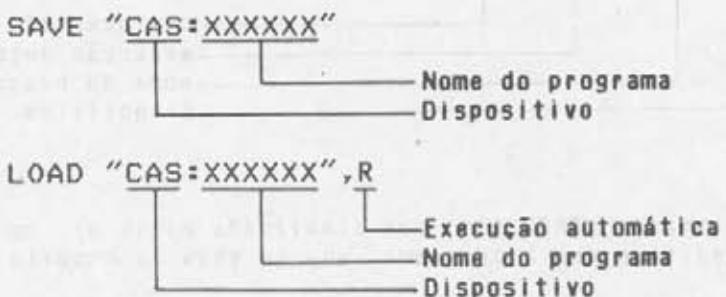
O comando CLOAD"nome" carrega da fita o programa gravado com CSAVE"nome". Digitando simplesmente CLOAD, o primeiro programa encontrado é lido.

SAVE

O comando SAVE tem quase a mesma finalidade que o CSAVE, entretanto ele serve para gravar programas em vários dispositivos (tela de texto, telas gráficas, disk-drive, gravador cassete, em modem, etc.).

Quando se usa SAVE para gravar em diskette, ele funciona de forma análoga ao CSAVE, isto é, copia o programa exatamente como ele está (com os tokens) na RAM. Quando usado para gravar em fita cassete, o SAVE é um pouco diferente. Neste caso o programa é gravado em fita na mesma forma em que é visualizado pelo comando LIST.

Ao gravar um programa em fita através do SAVE, o BASIC MSX lê cada linha do programa da RAM (que está condensado) e a transforma para o formato em que ela foi digitada (com códigos ASCII).

Além disso, o comando SAVE grava o programa em blocos de 255 bytes, em vez de gravá-lo de uma só vez.

Note que, quando você salva um programa longo com o SAVE ou carrega um programa com o LOAD, o micro liga e desliga várias vezes o gravador. Isso acontece porque com o SAVE os comandos do BASIC foram "expandidos" em um buffer (tal qual um arquivo).

Quando o buffer está lotado, ele é descarregado na fita e a operação continua até o fim do programa.

Com isso, são gravados muito mais bytes com o comando SAVE que com o comando CSAVE, ocupando muito mais fita e muito mais tempo.

A figura 6.3 apresenta o formato dos comandos SAVE e LOAD.

fig. 6.3 - Sintaxe dos comandos SAVE e LOAD.

```
SAVE "CAS:XXXXXX"
      |   |
      |   +-------------Nome do programa
      +-----------------Dispositivo

LOAD "CAS:XXXXXX",R
      |   |        |
      |   |        +----Execução automática
      |   +-------------Nome do programa
      +-----------------Dispositivo
```

Além disso, o comando SAVE permite gravar em diskette também no formato ASCII (como linhas "expandidas"), desde que se use a sintaxe:

```
SAVE "NOME",A
```

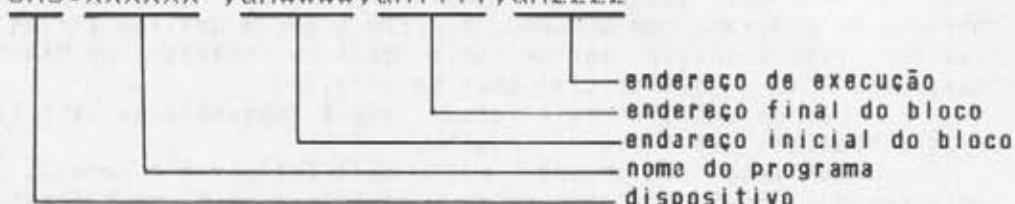
BSAVE

O comando BSAVE serve para quando nós temos um programa em linguagem de máquina no micro (uma sequência de bytes) e queremos gravá-lo em fita cassete (ou diskette).

Para fazermos isso, temos que informar ao micro o endereço de memória no qual o programa em linguagem de máquina está armazenado. Tanto o endereço inicial quanto o endereço final, para que o BASIC MSX saiba exatamente qual o bloco de memória que será "copiado" na fita.

Ao utilizarmos esse comando, podemos informar também qual o endereço em que deve começar a execução do programa, caso esse endereço não seja igual ao endereço inicial.

Para podermos recuperar um programa em linguagem de máquina que foi gravado em fita, nós utilizamos o comando BLOAD.

O endereço a partir do qual o programa em LM deverá estar, é especificado no momento da gravação, mas você pode deslocá-lo para frente ou para trás, bastando inserir após o nome do programa o deslocamento que se quer efetuar (o número deverá estar entre -32768 e 65535).

Se você desejar a execução automática do programa após o carregamento, basta colocar um "R" após o nome do programa.

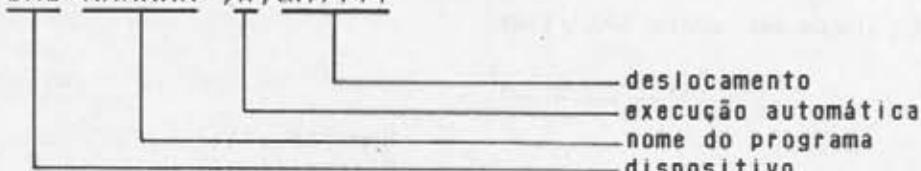
BSAVE, portanto, grava em fita uma "imagem" de um bloco da memória, sendo que esses dados são assumidos como um programa escrito em Linguagem de Máquina.

A figura 6.3b apresenta a sintaxe dos comandos BSAVE e BLOAD.

figura 6.3b sintaxe dos comandos BSAVE e BLOAD.

MERGE

O comando MERGE tem por finalidade misturar um programa que está num dispositivo com o programa que já está na memória do computador.

Esta "mistura" é muito útil quando queremos incluir uma subrotina de um programa em outro programa sem ter o trabalho de digitá-la novamente.

Apenas um detalhe deve ser observado: o comando MERGE só funciona se o programa gravado estiver no formato ASCII, ou seja, se tiver sido gravado através do comando SAVE. Se o programa a ser "emendado" foi salvo por um CSAVE, o MERGE não será efetuado!

Da mesma maneira que o SAVE utiliza um buffer para "expandir" as linhas do programa antes de gravá-lo, o MERGE o utiliza para guardar a linha do programa lida. Depois de lida uma linha, o MERGE "pensa" que você acabou de digitá-la e procura no programa que já está na memória o lugar correto onde a nova linha (lida do gravador) deverá ser colocada. Por isso, o MERGE precisa receber uma linha na mesma forma na qual ela seria digitada. E quem grava uma linha com essas características é o comando SAVE.

Já o comando CSAVE, por salvar o programa na forma condensada, faz com que o MERGE não entenda nada do que está sendo lido na fita.

Em resumo, utilize o comando CSAVE para gravar seus programas em BASIC por ser mais curto e mais rápido, mas se você precisar fazer um MERGE desses programas, regrave-os com o comando SAVE para que o MERGE consiga lê-los depois.

ORGANIZAÇÃO DOS DADOS NO CASSETE

Quando um comando de gravação em cassete é acionado, o programa ou o arquivo, são gravados em, no mínimo, dois blocos.

I - BLOCO-CABEÇALHO

Ao executar um comando de gravação no cassete, o MSX envia um sinal agudo, longo e contínuo (header), que servirá para sincronizar o micro com o gravador no momento da leitura. A seguir é enviado um cabeçalho composto de 16 bytes.

Os primeiros 10 bytes são iguais e indicam o comando pelo qual o programa ou os dados estão sendo armazenados.

A tabela da figura 6.4 indica o tipo de byte que é gravado por cada comando.

fig. 6.4 - Bytes de idetificação do modo gravação.

```
BSAVE     &HD0
CSAVE     &HD3
SAVE      &HEA
ARQUIVO   &HEA
```

Como você pode notar o byte de identificação do SAVE e de um arquivo é o mesmo, portanto você pode ler um programa gravado com o SAVE como se fosse um arquivo.

Os seis bytes seguintes ao bloco de identificação contém os códigos ASCII do nome do programa ou do arquivo.

Por exemplo, o seguinte cabeçalho da figura 6.5 indica que os dados do segundo bloco foram gravados pelo comando BSAVE e o nome do programa é ROTINA.

fig 6.5 - Exemplo de cabeçalho.

```
D0 D0 D0 D0 D0 D0 D0 D0 D0 D0  52 4F 54 49 4E 41
             |                  |  |  |  |  |  |
      tipo de gravação          R  O  T  I  N  A
```

Um erro que muitas vezes pode ocorrer, em uma fita contendo vários programas, e que esteja com o volume ou o azimute mal regulado é o aparecimento de um "programa" meio maluco. A listagem começa com vários tokens iguais e o computador não para de listar um programa sem sentido. Isso acontece porque o micro reconheceu o primeiro bloco (cabeçalho), mas não o segundo, que contém verdadeiramente o programa e interpretou o cabeçalho do seguinte como o segundo bloco do primeiro. Desta maneira o micro coloca na área de programa os bytes de identificação e como eles não tem o formato de um programa Sistema Operacional perde o controle obrigando até o desligamento do micro.

II - BLOCO-DADOS

O segundo bloco varia de formato de comando para comando.Vamos examinar cada um com detalhes.

CSAVE:A forma pelo qual o programa é gravado é a mesma que na memória RAM, ou seja, é mandado no segundo bloco, uma cópia fiel dos dados contidos na RAM, com tokens e tudo mais.

Se fosse possível enxergar os bytes gravados na fita, você veria que eles estão na mesma sequência que estavam na RAM quando o programa foi salvo.

Veja como um pequeno programa em BASIC seria gravado (figura 6.6).

fig. 6.6 - Forma de armazenamento do CSAVE.

```
10 PRINT
20 RUN
30 END
```

07	80	0A	00	91	00	0C	80	14	00	8A	00	13	80	1E	00	81	00	00	00
end. prox. linha		no. linha		PRINT		end. prox. linha		no. linha		RUN		end. prox. linha		no. linha		END	fim do progr.		

BSAVE: Os seis primeiros bytes do segundo bloco contém, na forma LSB e MSB, os endereços de início do programa em linguagem de máquina, o endereço final do programa e o endereço do início da execução do programa. Os bytes seguintes contém o programa em si. Veja o exemplo da figura 6.7

fig 6.7 - Gravação com o BSAVE.

00	80	60	C0	00	C0	00	FF	D3	E5	49	1.	..	..	...
end. inicial		end. final		end exec.			programa em LM							

SAVE: O comando SAVE, dependendo da quantidade de bytes a serem gravados, não apresenta um bloco de dados, mas vários, porque o SAVE, no cassete, tem o funcionamento idêntico ao de um arquivo. Com ele são gravados vários blocos de 256 bytes.

As linhas ficam armazenadas com o respectivo código ASCII seguido dos bytes &H0D (RETURN) e &H0A (salta linha).

Com o comando SAVE, na fita, o programa BASIC exemplificado na figura 6.6 seria gravado da forma indicada na figura 6.8 .

fig 6.8 - gravação com o SAVE

31	30	20	50	52	49	4E	54	0D	0A	32	30	20	52	55	4E	0D	0A	33	30	20	45	4E	44
1	0		P	R	I	N	T			2	0		R	U	N			3	0		E	N	D

Mesmo que um bloco não seja totalmente preenchido, o micro grava 256 bytes na fita. Os bytes que não contém dados são preenchidos com 1A.

ARQUIVOS: Os dados em arquivo contêm códigos ASCII, mas existem alguns parâmetros de gravação.

O comando PRINT#, que guarda os dados no buffer, tem um comportamento para dados numéricos e outro para os alfanuméricos.

Cada comando PRINT# que é executado armazena no buffer, depois dos dados, os bytes &H0D e &H0A.

Os dados numéricos são gravados com um espaço (caracter &H20) antes e um depois.

Se houver uma vírgula entre dois dados em um mesmo PRINT#, é gravado o número de espaços que seriam impressos na tela caso fosse usado o comando PRINT.

Se o seguinte PRINT# fosse executado, em um arquivo já aberto, os dados seriam armazenados primeiramente no buffer e posteriormente na fita da seguinte forma:

```
PRINT#1,"Aleph";1;2
```

41	6C	65	70	68	20	31	20	20	32	20	0D	0A
A	l	e	p	h		1			2			

O programa a seguir é um leitor de cabeçalho de programas em fita. Ele utiliza três rotinas do BIOS (veja apêndice IV) TAPION, TAPIN e TAPIOF para sincronizar a leitura do gravador, ler os dados da fita e parar a leitura, respectivamente.

fig.6.10 - Programa HEADER.

```
10 DATA CD,E1,00,D8,21,00,C0,06,10,E5,C
5,CD,E4,00,C1,E1,D8,77,23,10,F4,CD,E1,0
0,D8,21,10,C0,06,06,E5,C5,CD,E4,00,C1,E
1,D8,77,23,10,F4,C9

20 FOR L=0 TO 42:READ A$:POKE 60000!+L,
VAL("&H"+A$):NEXT L:DEFUSR=60000!:DEF F
NPE(X)=PEEK(X)+256*PEEK(X+1)
30 CLS:COLOR 15,4:PRINT:PRINT:PRINT"COL
OQUE A FITA PARA LER E APERTE <CR>":PRI
NT:PRINT
40 I$=INPUT$(1):X=USR(0):MOTOR OFF
50 PRINT:PRINT "NOME........ ";:FOR L=4
9162! TO 49167!:PRINT CHR$(PEEK(L));:NE
XT L:A=PEEK(49152!):PRINT:PRINT:PRINT"F
ORMATO..... ";
51 IF A=208 THEN PRINT "BINARIO":PRINT:
PRINT "INICIO...... ";HEX$(FNPE(49168!)
):PRINT:PRINT "FIM......... ";HEX$(FNPE
(49170!)):PRINT:PRINT "ENTRADA..... ";H
EX$(FNPE(49172!))
52 IF A=211 THEN PRINT "BASIC"
53 IF A=234 THEN PRINT "ASCII"
54 IF A<>208 AND A<>211 AND A<>234 THEN
 PRINT"erro de leitura"
60 PRINT:PRINT:PRINT "APERTE QUALQUER T
ECLA PARA REINICIAR":I$=INPUT$(1):GOTO
30
```

Disk-drive

Você pode conectar ao MSX vários periféricos, entre eles o disk-drive. Disk-drive é o nome dado ao equipamento que grava e lê dados em discos magnéticos flexíveis.

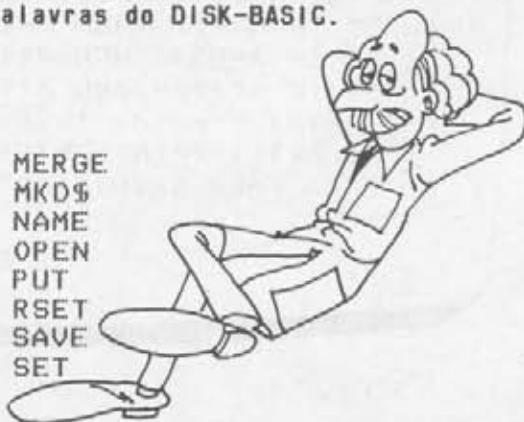
Os discos nos quais o drive lê e grava informações são constituidos de material semelhante ao das fitas cassete, só que no formato de um disco. Por serem maleáveis, são também conhecidos como discos flexíveis (mesmo quando fechados num envelope rígido).

Existem vários tamanhos de discos flexíveis no mercado (mais conhecidos como diskettes): 5 1/4", 8", 3 1/2", 3", etc..

Para cada tamanho de diskette existe um drive especifico para ler e gravar informações, cada um com suas própias características. É muito comum, portanto, dizer:"o drive de oito polegadas..." ou "o drive de 5 e 1/4 de polegadas", que na verdade quer dizer "o drive feito para o disco de 8 polegadas", e assim por diante.

A finalidade de um drive é a mesma de um gravador cassete ligado ao micro: armazenar dados, só que com muito mais vantagens. O drive é muito mais confiável que um gravador cassete; é muito mais rápido na transferência de dados entre ele (o drive) e o computador; permite acessar uma certa informação sem ter que ler todas as informaçoes presentes antes daquelas que queremos (em uma fita cassete para lermos um programa no meio da fita, por exemplo, temos que passar por todos os programas que estão desde o começo da fita, até achar aquele que queremos, o que já não acontece com o drive).

Você já viu, no item anterior, como funcionam os comandos SAVE, LOAD, OPEN, e MERGE. Na verdade, com um disk-drive conectado ao micro, existem muitos outros recursos disponíveis. Quando conectamos um disk-driver ao MSX, além de acionarmos mais um periférico, acordamos dois gigantes adormecidos: o DISK BASIC e o MSX-DOS . Vamos estudar um pouco esses dois poderosos recursos do padrão MSX.

DISK-BASIC

O disk-driver permite ao BASIC MSX acessar arquivos em disco que não eram possíveis de serem acessados em fita cassete. Isto pelo fato do cassete só permitir leitura sequencial, ou seja, você só pode acessar um dado após ter lido o dado que o antecede.

Com base nisso, você pode, por exemplo, criar arquivos que contenham os nomes de todos os seus amigos, com seus respectivos endereços e telefones. Quando você quiser saber o telefone de alguém, bastará digitar o começo do nome e, em poucos segundos, você terá todos os dados no vídeo: nome completo, endereço, telefone, etc...

Os vários comandos acrescentados ao BASIC e que são exclusivos para acesso a disco constituem um sub-conjunto chamado DISK-BASIC. Na figura 6.11 podemos observar todas as palavras do DISK-BASIC.

fig. 6.11 - Palavras reservadas do DISK-BASIC.

BLOAD	CVI	IPL	MERGE
BSAVE	CVS	KILL	MKD$
CLOAD	DSKI$	LFILES	NAME
CLOSE	EOF	LOAD	OPEN
COPY	FIELD	LOC	PUT
CSAVE	FILES	LOF	RSET
CMD	FORMAT	LSET	SAVE
CVD	GET	MAX	SET

Algumas dessas palavras você já conhece. Elas foram "melhorados" e com o DISK BASIC você pode usá-los com o disco em vez de usá-los apenas com o cassete ou impressora.

130

Com o DISK-BASIC pode-se fazer muito mais coisas do que a princípio pode parecer. Verdadeiros sistemas podem ser desenvolvidos, como por exemplo:

- Mala Direta;
- Contas a pagar/receber;
- Contabilidade;
- Folha de Pagamento;
- etc ...

O DISK-BASIC, em resumo, é o mesmo BASIC que você já conhece, porém, com alguns comando e funções a mais. Apenas isso basta para caracterizar qualquer MSX como um micro potencialmente profissional, além de pessoal.

MSX-DOS

O MSX-DOS é um Sistema Operacional muito poderoso e versátil. D.O.S. é a abreviação de DISK OPERATING SYSTEM (Sistema Operacional em Disco).

Um Sistema Operacional é um conjunto de instruções e de funções que serve de "mediador" entre o hardware do micro (circuito eletrônico) e os programas (software) a serem usados. O Sistema Operacional controla todas as operações envolvendo periféricos (teclado, vídeo, impressora, o próprio disk-drive , etc...). Ele gerencia, também, todos os arquivos em disco, organizando o espaço por eles ocupado e servindo de "comunicador" com o usuário durante a execução do programa.

Quando você utiliza o MSX-DOS, você não tem nada na memória do micro que lhe permita fazer outra coisa além de copiar arquivos de um periférico para outro, listar programas, etc.

Com o MSX-DOS, você pode chamar o BASIC, entrando assim no DISK-BASIC e, posteriormente, voltar ao MSX-DOS.

Além disso tudo, o MSX-DOS foi elaborado de maneira a ser compatível com o CP/M , que é o Sistema Operacional com mais programas já desenvolvidos em todo o mundo. Isso faz com que o MSX-DOS, excelente por si só, já venha com uma enorme bagagem de software disponível no mercado.

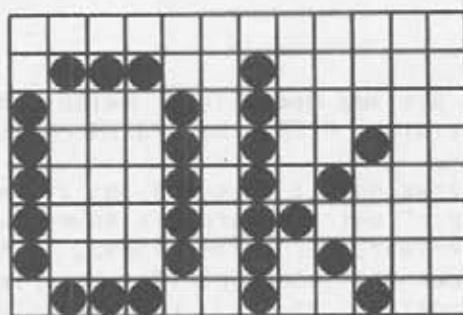
Impressora

Impressora é um periférico que tem como finalidade passar para o papel uma dada informação que está na memória do micro.

Ela é muito semelhante à uma máquina de escrever, só que normalmente não tem o teclado: afinal quem manda as informações a serem escritas é o micro e não uma pessoa, como em uma máquina de escrever.

Existem no mercado, hoje em dia, várias marcas e modelos de impressoras que podem ser conectadas ao MSX, cada uma com suas características próprias. Algumas são mais adequadas a certas aplicações do que outras.

Existem basicamente 4 tipos de impressoras:

1) impressora matricial (ou serial);
2) impressora de linha (ou paralela);
3) impressora térmica ;
4) impressora laser.

Vamos nos deter apenas nas impressoras matriciais por serem as mais usadas em microcomputadores, devido ao seu baixo custo e boa qualidade de impressão. As impressoras de linha e a laser são de alta velocidade de impressão (às vezes mais rápidas que o próprio micro) tornando-as caras e ociosas quando ligadas a um microcomputador, e as impressoras térmicas são muito lentas e não se enquadram numa faixa mais profissional como os micros MSX.

O nome matricial ou serial vem do fato dela imprimir um caractere através de uma matriz de pontos. Este processo é análogo ao modo com o qual são gerados os caracteres no vídeo do MSX.

Essa matriz de pontos é formada por uma coluna de agulhas (geralmente 9) dispostas na vertical sobre um carro que se movimenta no sentido horizontal.

A medida em que esse carro vai se movendo, as agulhas vão sendo acionadas de acordo com o caractere a ser impresso. Veja na figura 6.12 como normalmente aparece a mensagem Ok nessas impressoras:

fig. 6.12 - Impressão do Ok em impressoras matriciais.

Algumas impressoras são também chamadas de impressoras gráficas, por possuirem recursos de maneira a permitir a impressão de desenhos de vários tipos, além de letras e números.

Outras conseguem imprimir letras e números com aparência semelhante a das letras de uma máquina de escrever comum (near letter quality) e existem algumas que escrevem letras idênticas àquelas impressas por máquina de escrever (letter quality).

Se você pretende ter uma impressora apenas para listar os seus programas e para obter relatórios impressos de maneira geral, não é preciso que ela seja gráfica, ou do tipo "qualidade de carta", pois estas últimas, devido a esses recursos adicionais, são mais caras. Mas se você deseja uma impressora também para obter desenhos e gráficos impressos, você terá obrigatoriamente que adquirir uma impressora gráfica.

Outra característica que deve ser levada em conta, é a velocidade de impressão.

Essa velocidade de impressão é o tempo que a impressora leva para imprimir um certo número de caracteres por segundo.

As impressoras que são encontradas no mercado têm geralmente uma velocidade de apoximadamente 100 CPS (100 caracteres em um segundo), quando estão imprimindo textos normais.

Se uma impressora com "qualidade de carta" estiver imprimindo um texto nesta modalidade (ou seja, as letras se assemelham àquelas de uma máquina de escrever) a velocidade de impressão é bem menor: aproximadamente 18 a 20 CPS.

Uma impressora gráfica, por sua vez, quando está fazendo um gráfico ou um desenho, pode demorar vários minutos até completá-lo.

Outra característica : quantidade de caracteres que podem ser impressos em uma linha.

Normalmente são impressos 80 ou 132 caracteres em uma linha.

Uma impressora de 80 caracteres é menor e mais barata do que uma que consegue imprimir 132 caracteres em uma mesma linha.

Por fim, a principal característica: a comunicação com o micro. Essa comunicação é a maneira pela qual a impressora e o micro trocam informações e pode ser serial ou paralela.

O micro MSX já tem incorporado internamente um circuito eletrônico (interface) para se comunicar (transmitir dados) com a impressora, com o formato paralelo, padrão "centronics". Portanto, quando você for adquirir uma impressora, antes de mais nada você deve certificar-se de que ela possui comunicação paralela com este mesmo padrão, caso contrário você não conseguirá conectá-la ao micro!

Uma vez escolhida a impressora, vamos ver como utilizá-la.

Existem no BASIC MSX dois comandos ligados diretamente com o uso da impressora. São eles: LPRINT e LLIST.

Estes dois comandos funcionam de maneira semelhante ao PRINT e LIST, só que usando a impressora. Portanto, tudo o que vale para o PRINT e o LIST, vale também para o LPRINT e o LLIST.

Outra maneira de se enviar dados para impressão é através dos OPEN e PRINT #. Vamos ver como.

Inicialmente, devemos abrir um arquivo como saída utilizando o comando OPEN e informando LPT (impressora) como sendo o periférico que irá receber os dados:

```
OPEN "LPT:" FOR OUTPUT AS #1
```

Depois, sempre que quisermos enviar dados para a impressora, utilizamos o comando PRINT #1 .

O resultado final, utilizando LPRINT ou utilizando um arquivo, é o mesmo, não havendo uma preferência quanto ao uso de um ou outro comando.

Vamos a alguns programinhas para exemplificar.

Na ROM nós temos definidos todos os 255 caracteres do MSX.

O programa da figura 6.13 espera você digitar uma mensagem, pesquisa na ROM o "formato" dos caracteres que a compõem e imprime na impressora. (Note que o endereço pesquisado na linha 140 é obtido pela leitura dos bytes 4 e 5 da ROM - veja capítulo 2).

fig. 6.13 - Programa para imprimir ampliado.

```
100 REM      AMPLIADOR
110 BR$ = " "
120 PR$ = "#"
130 OPEN "lpt:" FOR OUTPUT AS #1
140 ET = PEEK(4) + 256 * PEEK(5)
150 PRINT#1,CHR$(27);CHR$(65);CHR$(8)
160 INPUT "Qual a mensagem ";CP$
170 FOR RW% = 0 TO 7
180  FOR NV% = 1 TO LEN(PR$)
190   FOR I% = 1 TO LEN(CP$)
200    AC% = ASC(MID$(CP$,I%,1))
210    IF AC% <> 1 THEN 240
220    I% = I% + 1
230    AC% = ASC(MID$(CP$,I%,1)) - 64
```

```
240    BT$ = BIN$(PEEK(AC%*8+ET+RW%))
250    BT$ = RIGHT$("0000000"+BT$,8)
260    FOR J% = 1 TO 8
270     X$ = MID$(BT$,J%,1)
280     IF X$ = "0" THEN PRINT #1,BR$;
290     IF X$ = "1" THEN PRINT #1,PR$;
300    NEXT J%
310   NEXT I%
320   PRINT #1," "
330  NEXT NV%
340 NEXT RW%
350 GOTO 160
Ok
■
```

Se você não tiver uma impressora, altere a linha 130 para:

```
130 OPEN "CRT:" FOR OUTPUT AS #1
```

e você terá o mesmo resultado obtido na impressora em seu vídeo.

Experimente, também, substituir as linhas 110 e 120 por:

```
110 BR$="**"
120 PR$="  "
```

e veja o efeito!

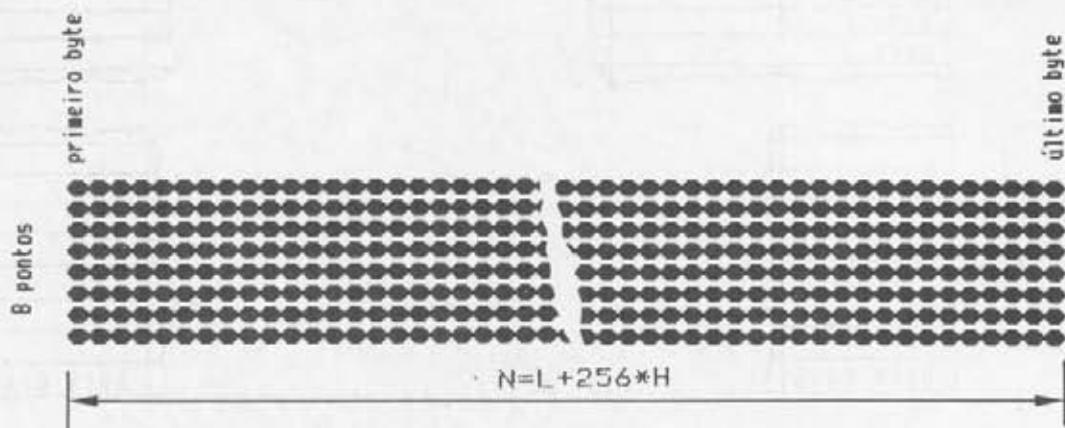
Vimos, portanto, que usando a impressora com um arquivo (através dos comandos OPEN e PRINT #1) fica muito fácil redirecionarmos os dados de saída, alterando apenas uma linha do programa para enviarmos os dados para outro periférico.

Até agora, o micro só mandou os códigos de cada letra, e a própria impressora se encarregou de imprimir a letra ponto a ponto, de acordo com um padrão previamente definido quando da fabricação da impressora, ou seja, quando mandamos imprimir a letra "A" (maiúscula) o micro só mandou um byte para a impressora: o 41, que é o código ASCII (em hexadecimal) da letra A.

As impressoras gráficas, porém, permitem acessar cada agulha de impressão individualmente, podendo assim imprimir qualquer coisa que imaginarmos.

Para que isso ocorra, é necessário enviar uma sequência tal de caracteres para a impressora de maneira que ela saiba que a partir daquele momento serão enviados não os códigos ASCII, mas sim os códigos que mostram quais os pontinhos que queremos que ela imprima.

Vamos supor que sua impressora, como a maioria das impressora nacionais, entre no modo gráfico segundo o padrão EPSON.

A impressora entra no modo gráfico através do seguinte comando:

a - indica, à impressora, que vão chegar códigos de controle;
b - coloca a impressora no modo gráfico;
c e d - delimitam o número N de bytes que serão impressos com gráficos, segundo a relação:

N = L + 256 * H

O valor de N tem um limite máximo em função do número de caracteres que cabem numa linha de impressão.

fig. 6.14 - Números de colunas gráficas numa linha.

Experimente, por exemplo, digitar o programa da figura 6.15a.

fig. 6.15a - Xadrez.

```
10 LPRINT CHR$(27);CHR$(75);CHR$(14);CH
R$(1);
20 FOR I = 1 TO 135
30     LPRINT CHR$(&B10101010);CHR$(&B0
1010101);
40 NEXT I
```

A linha 10 coloca a impressora em modo gráfico para imprimir:

14 + 256 * 1 = 270 bytes

A linha 30 envia dois bytes que formam um xadrez pois os 1 e 0 se alternam.

As linhas 20 e 40 fazem esses dois bytes serem enviados 135 vezes pois:

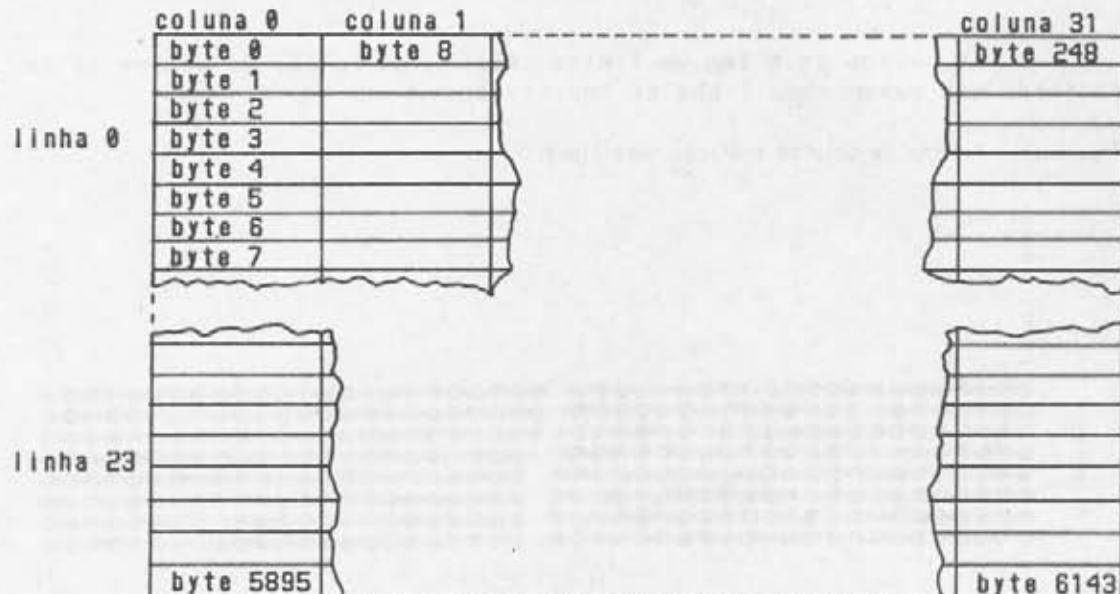
2 * 135 = 270

Você obterá a faixa xadrez da figura 6.15b .

fig. 15b - Resultado do programa Xadrez.

Para copiar a tela gráfica basta lê-la byte a byte e enviar estas informações para a impressora. Como já vimos no capítulo 4, na SCREEN 2 os bytes estão armazenados conforme a figura 6.16 .

fig. 6.16 - Organização da SCREEN 2 na VRAM.

Como os bytes da tela estão organizados na horizontal e os da impressora na vertical, é conveniente lê-la começando do byte 5895 até o byte 0 para a primeira coluna e assim sucessivamente.

Na última coluna começaremos pelo 6143, terminando no 248.

Vamos então digitar um programa que gere um desenho na tela de alta resolução e copiá-lo com uma sub-rotina adequada.

Obviamente a figura sairá "deitada" na impressora.

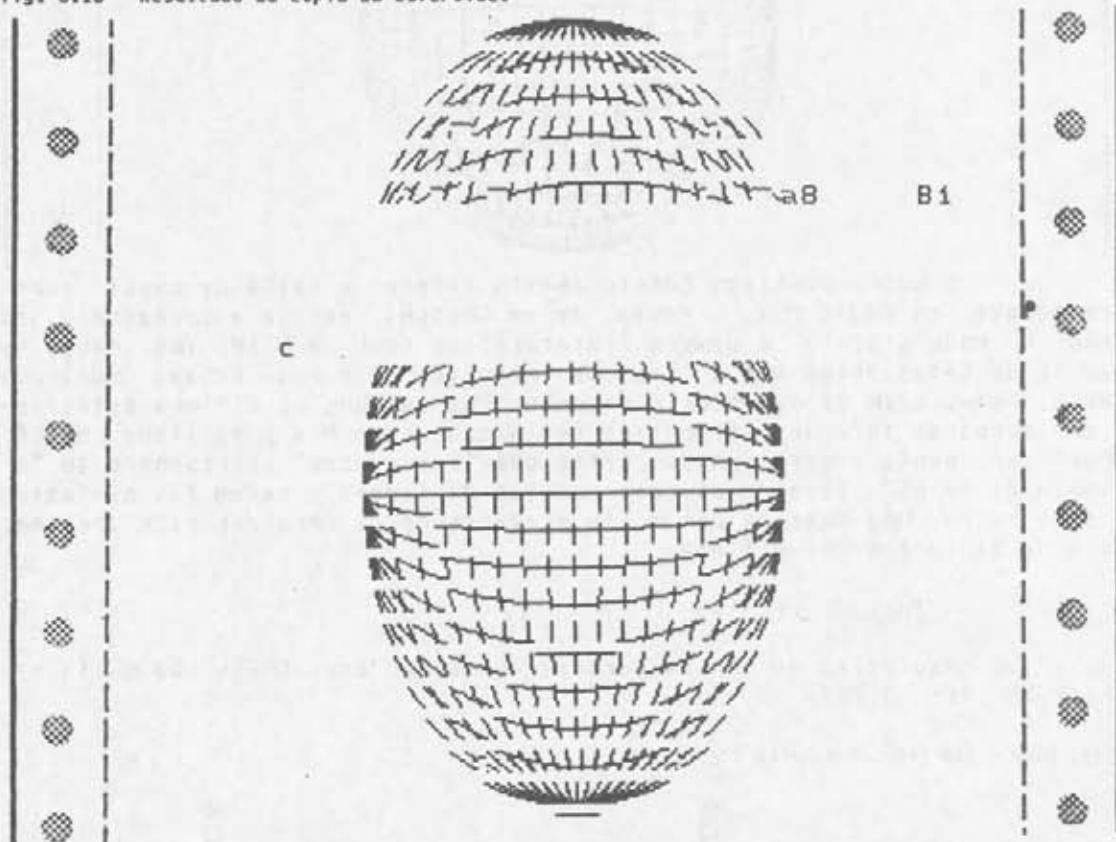
Digite o programa da figura 6.17.

fig. 6.17 - Cópia de esferóide.

```
10 COLOR1,7,7:SCREEN2:FORB=80TO1STEP-10
:CIRCLE(128,80),80,1,,,80/B:CIRCLE(128,
80),80,1,,,B/80:NEXTB:LINE(128,160)-(12
8,0):LINE(48,80)-(208,80)
20 GOSUB 50000
30 END
50000 REM ----- copy ------
50040 FOR C=0 TO 31
50060 LPRINT CHR$(27);CHR$(75);CHR$(192
);CHR$(0);
50080 FOR L=23 TO 0 STEP -1
50100 FOR X=7 TO 0 STEP -1
50120 U=VPEEK((C*8+256*L)+X)
50140 LPRINT CHR$(U);
50160 NEXT X
```

```
50180 NEXT L
50200 LPRINT CHR$(10);
50220 NEXT C
50260 LPRINT CHR$(27);CHR$(81);CHR$(39)
50280 RETURN
```

A linha 10 gera uma figura, e a sub-rotina 50000 vai copiá-la na impressora.

Nesta sub-rotina o laço 50040-50220 copia as 32 colunas da esquerda para a direita. Dentro do laço, a linha 50060 prepara a impressora para receber 192 bytes(24 linhas de 8 bytes cada - veja figura 6.16). Os laços 50080-50180 e 50100-50160 copiam as 24 linhas de 8 bytes cada, de baixo para cima.

A linha 50140 envia cada byte à impressora.

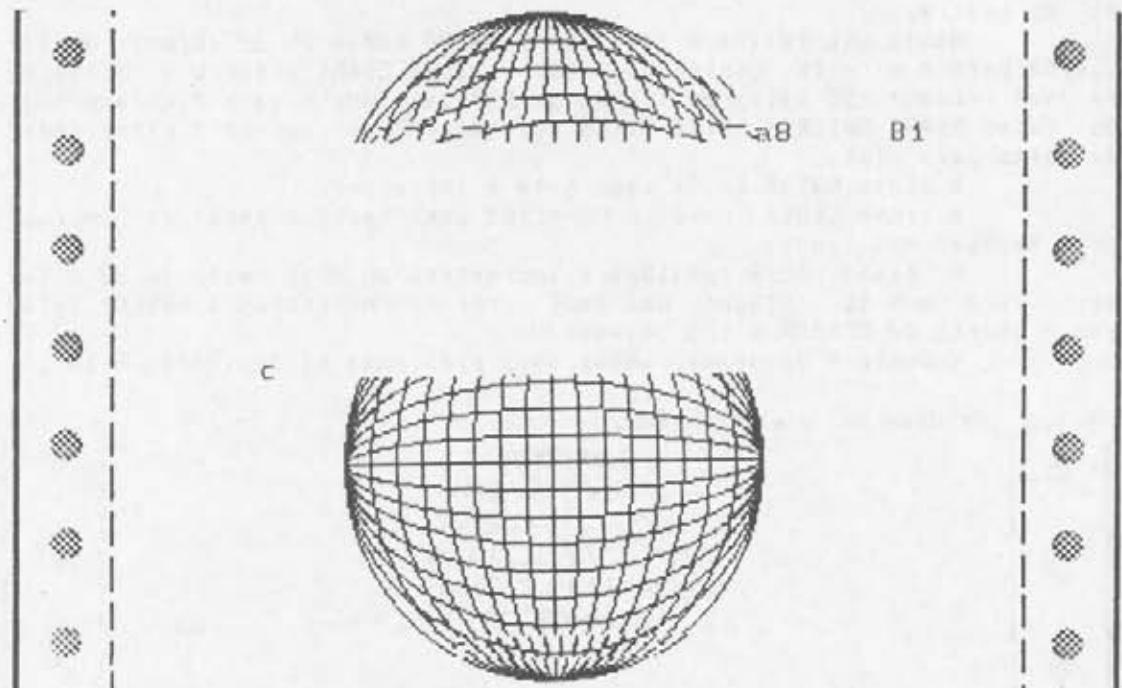
A linha 50200 envia o CHR$(10) para fazer o papel da impressora avançar uma linha.

A linha 50260 recoloca a impressora no modo texto de 39 colunas , para que as listagens que você tirar dos programas a seguir saiam com o padrão do SCREEN 0 (39 colunas).

Rodando o programa, vemos dois problemas sérios (fig. 6.18).

fig. 6.18 - Resultado da cópia do esferóide.

O primeiro pode ser facilmente consertado: para que não apareçam as faixas de entrelinhamento basta avançar a linha de 8 em 8 e não de 13 em 13 pontos. Para isso basta acrescentar a linha:

```
50020 LPRINT CHR$(27);CHR$(65);CHR$(8)
```

para reduzir o avanço para 8 pontos e:

```
50240 LPRINT CHR$(27);CHR$(65);CHR$(13)
```

para voltar ao normal uma vez terminada a cópia.

Rodando o programa assim alterado , vemos que o primeiro problema foi eliminado (fig. 6.19).

fig. 6.19 - Cópia do esferóide parcialmente consertado.

O outro problema (deslocamento lateral e salto do papel) ocorre porque, no BASIC MSX, o envio de um CHR$(9), esteja a impressora ou não no modo gráfico, é sempre interpretado como um HTAB, implicando no envio de bytes nulos até a próxima impressão cair numa coluna múltipla de 8. Isto, além de deslocar o desenho, faz com que os últimos bytes sejam impressos fora dos parâmetros definidos pelo H e L da linha 50060. Por azar, nesta figura, um dos bytes que "extravazam" corresponde ao "avanço de folha", fazendo os dois pedaços da figura ficarem tão distantes um do outro. Uma maneira não muito elegante de se resolver este problema é a de se introduzir a linha:

```
50130 IF U=9 THEN U=17
```

pois as sequências de pontos geradas por estes dois CHR$s são muito parecidas (fig. 6.20).

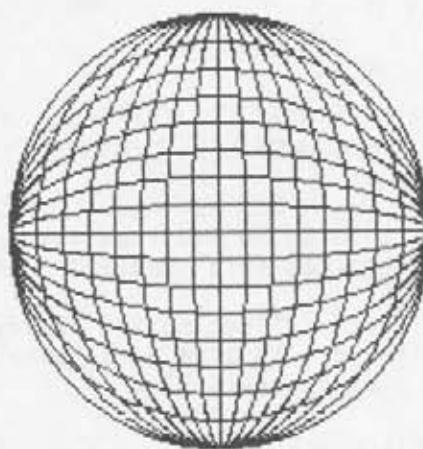



fig. 6.20 - Impressão dos bytes 9 e 17.

Rodando o programa com essa alteração você obterá a figura 6.21.

fig. 6.21 - Cópia do esferóide sem "galhos".

Obviamente, existe uma solução mais "limpa": no BIOS da ROM (veja apêndice IV) há uma sub-rotina, LPTOUT, que escreve na impressora sem que haja o problema do CHR$(9):

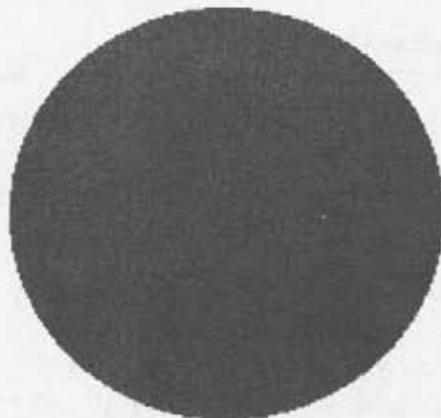
```
LD A.00      3E 00       carrega o 00 no acumulador
CALL 00A5    CD A5 00    chama o LPTOUT
RET          C9          volta ao BASIC
```

Vamos alterar o programa de maneira a usar essa sub-rotina: isto é feito introduzindo-se as linhas 50010, 50012 e mudando as linhas 50130 e 50140 (fig. 6.22).

fig. 6.22 - Cópia com sub-rotina em LM.

```
10 COLOR1,7,7:SCREEN2:FORB=80TO1STEP-10
:CIRCLE(128,80),80,1,,,80/B:CIRCLE(128,
80),80,1,,,B/80:NEXTB:LINE(128,160)-(12
8,0):LINE(48,80)-(208,80)
20 GOSUB 50000
30 END
50000 REM ----- copy ------
50010 DATA 3E,00,CD,A5,00,C9
50012 FOR I = 0 TO 5 : READ A$:POKE 510
00!+I,VAL("&H"+A$): NEXT I : DEFUSR0=51
000!
50020 LPRINT CHR$(27);CHR$(65);CHR$(8)
50040 FOR C=0 TO 31
50060 LPRINT CHR$(27);CHR$(75);CHR$(192
);CHR$(0);
50080 FOR L=23 TO 0 STEP -1
50100 FOR X=7 TO 0 STEP -1
50120 U=VPEEK((C*8+256*L)+X)
50130 POKE 51001!,U
```

```
50140 Z=USR0(0)
50160 NEXT X
50180 NEXT L
50200 LPRINT CHR$(10);
50220 NEXT C
50240 LPRINT CHR$(27);CHR$(65);CHR$(13)
50260 LPRINT CHR$(27);CHR$(81);CHR$(39)
50280 RETURN
```

Como você pode notar, o programa funciona sem que haja necessidade de fazer a substituição pelo CHR$(17); deselegante,apesar de quase imperceptível!

Não vamos, porém, ficar muito felizes, pois existem outros problemas. Mude as linhas de 10 a 30 para gerar um círculo pintado de vermelho na tela como o da figura 6.23 .

```
10 SCREEN 2
20 CIRCLE (128,96),80,8:PAINT(128,96),8
30 GOSUB 50000: end
```

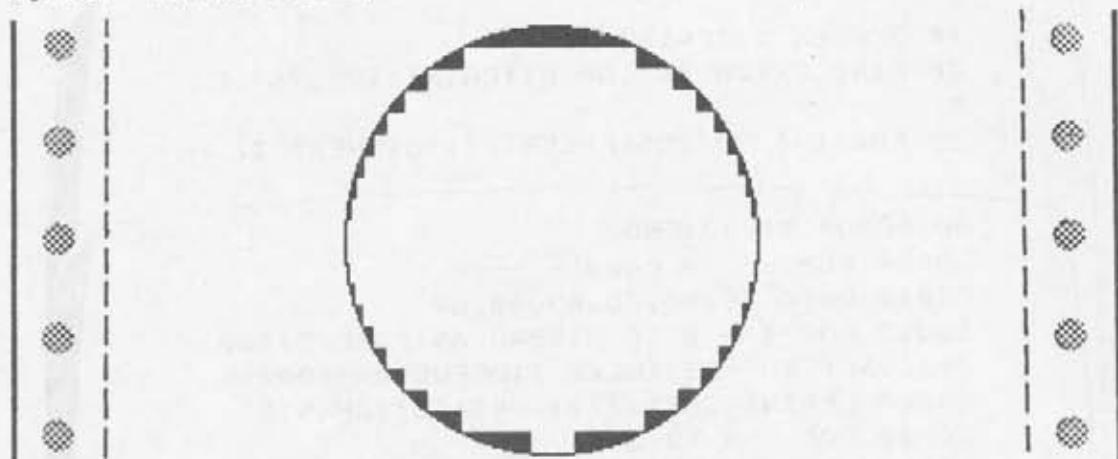



fig. 6.23 - Banzai!!

Rode o programa e veja como o PAINT não acende os pontos na tela, mas sim muda o que é cor de frente e cor de fundo no arquivo de

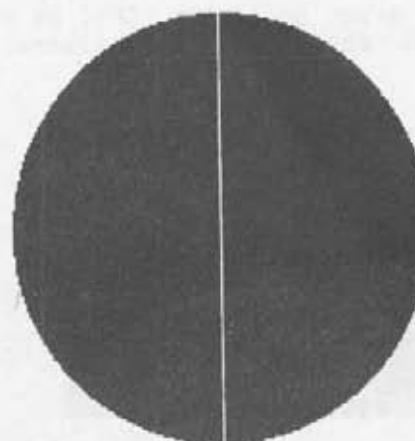
atributos (fig. 6.24).

fig. 6.24 - Problema produzido pelo PAINT.

Para eliminar este inconveniente devemos ler, também, o arquivo de atributos: se a cor de fundo for igual à cor que havia sido definida como de frente, devemos acender todos os 8 pontos que estavam apagados no arquivo de tela.

Isso é feito acrescentando-se as linhas 50122 e 50125:

```
50122 V=VPEEK((C*8+256*L)+X+8192)
50125 IF V MOD 16 = 8 THEN U = 255
```

Rode o programa assim alterado e você verá como esse inconveniente foi eliminado.

Vamos agora sofisticar um pouco nossa figura para ver se podemos copiá-la sem problemas.

Podemos chamar a cor de fundo de CP (cor do papel) e a cor de fronte de CT (cor da tinta).

Usando CP=1 (preto) e CT=8 (vermelho), podemos alterar as linhas 10, 20 e 50125.

Acrescentando a linha 25 podemos fazer o círculo ficar cortado por uma linha horizontal (na impressora ela será vertical), como na figura 6.25 .

fig. 6.25 - Banzai cortado.

fig. 6.26 - Programa de cópia alterado.

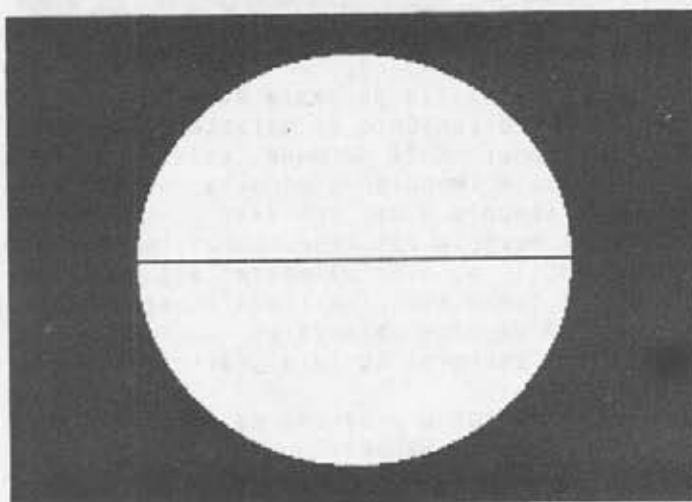
```
10 SCREEN 2:CP=1:CT=8
20 CIRCLE(128,96),80,CT:PAINT(128,96),C
T
25 FOR I=0 TO 255:PRESET(I,96):NEXT I

30 GOSUB 50000:END
50000 REM ----- copy ------
50010 DATA 3E,00,CD,A5,00,C9
50012 FOR I = 0 TO 5:READ A$:POKE 51000
!+I,VAL("&H"+A$):NEXT I:DEFUSR0=51000!
50020 LPRINT CHR$(27);CHR$(65);CHR$(8)
50040 FOR C=0 TO 31
50060 LPRINT CHR$(27);CHR$(75);CHR$(192
);CHR$(0);
50080 FOR L=23 TO 0 STEP -1
50100 FOR X=7 TO 0 STEP -1
50120 U=VPEEK((C*8+256*L)+X)
50122 V=VPEEK((C*8+256*L)+X+8192)
50125 IF V MOD 16 =CT  THEN U=255
50130 POKE 51001!,U
50140 Z=USR0(0)
50160 NEXT X
50180 NEXT L
50200 LPRINT CHR$(10);
50220 NEXT C
50240 LPRINT CHR$(27);CHR$(65);CHR$(13)
50260 LPRINT CHR$(27);CHR$(81);CHR$(39)
50280 RETURN
```

Rode o programa assim modificado (fig. 6.26) e você verá surgir mais um problema: a linha traçada no meio do círculo não implica no apagamento de pontos no arquivo da tela. Ela produz uma alteração no arquivo atributos, alterando a cor de fundo (CP). Se o risco estiver num byte em que nem todos os 8 pontos estejam afetados, ele aparece no arquivo de tela (fig. 6.27).

fig. 6.27 - Problema no Banzai cortado.

Quando todos os 8 pontos são afetados, assim como acontecia no PAINT, quem se encarrega de enviá-lo para o vídeo é o arquivo de atributos.

Para resolver mais esse problema, devemos acrescentar a linha 50128 ao nosso programa:

```
50128 IF V\16=CP THEN U=0
```

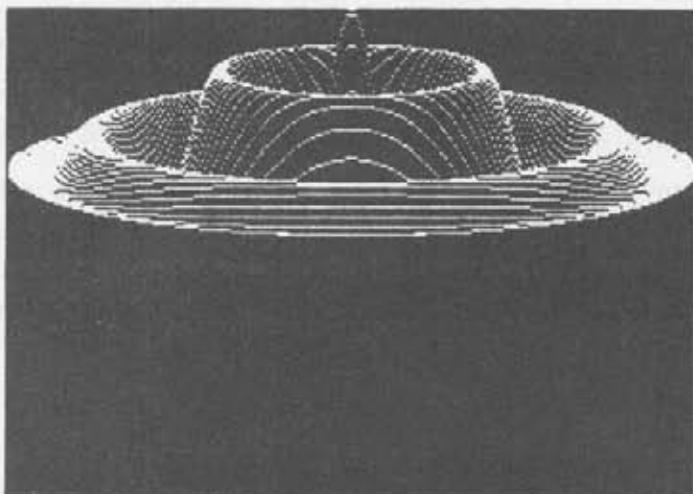
Rodando o programa, ele copia tudo a que temos direito!

Se quisermos inverter a figura (negativo) basta mudar os 0 por 1 e os 1 por 0 na notação binária do byte lido por U. Isto é mais simples do que parece: basta substituir o U por 255-U .

Faça esta alteração na linha 50130 e você obterá a figura 6.28.

fig. 6.28 - Banzai cortado em negativo.

Se você tiver bastante paciência, substitua as linhas de 10 a 30 do programa da figura 6.26 por:

```
10 COLOR15,1,1:SCREEN2:FORB=0TO127:X4=B
*B:M=-128:A=SQR(16384-X4):FORI=-ATOASTE
P3:R=SQR(X4+I*I)/128:F=COS(16*R)*(1-R)*
2:Y=I/5+F*32:IFY<=MTHEN20ELSEM=Y:Y=128+
Y:X=128+B:PSET(X,191-Y):X=128-B:PSET(X,
191-Y)
20 IFP=1THEN20ELSENEXTI,B:P=1:CT=15:CP=
1:GOSUB50000:END
```

e a linha 50130 por:

```
50130 POKE 51001!,255-U
```

e você obterá uma figura que mostra muito bem as potencialidades da alta resolução do MSX (figura 6.29).

fig. 6.29 - Cogumelo em negativo.

Uma outra grande utilidade desta sub-rotina é a de se escrever na impressora gráfica com o conjunto de caracteres do MSX.

Quando a impressora está no modo texto, o computador envia códigos que servem para que a impressora escolha, no seu arquivo de caracteres, qual a letra ou símbolo a ser impresso.

A esmagadora maioria das impressoras tem um arquivo de caracteres com os códigos ASCII e, eventualmente, alguns símbolos gráficos. Porém, a menos que ela tenha sido fabricada especialmente para o padrão MSX, ela não terá muitos de seus caracteres.

A solução será escrever na tela gráfica e depois copiá-la byte a byte com a rotina COPY.

Experimente digitar o programa da figura 6.31 e rode-o junto com a sub-rotina COPY (50000 a 50280).

fig. 6.30 - Escrevendo em MSX numa impressora não-MSX.

```
10 SCREEN 2: OPEN "GRP:" FOR OUT
PUT AS #1
15 PRESET (20,0):PRINT#1,"◆◆◆◆◆◆
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆"
20 PRESET (20,20):PRINT #1,"Exem
plo de impressão MSX "
30 PRESET (20,40):PRINT #1,"APKI
MEΔEΣ"
40 PRESET (20,60):PRINT#1,"ROSÆ
SUNT PULCHRÆ"
50 PRESET (20,80):PRINT#1,"☼ ☻ ♫
 ☺ ♣ ♠ ♦ ♥ * ╳ ♂ ♀"
60 PRESET (20,100):PRINT#1,"¶ ¥
√ γ δ ∈ θ Γ ø ₮ Ω ∞"
70 PRESET (20,120):PRINT#1," ⌠""

80 PRESET (19,128):PRINT#1," | s
in(2¶+ωt)² dt = ¿¿"
90 PRESET (20,136):PRINT#1," ⌡"
100 PRESET (20,160):PRINT#1,"◆◆◆
◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆"
110 GOSUB 50000:END
```

Obs.: Acrescente a seguir a sub-rotina de 50000 a 50280 da figura 6.26.

Você obterá (obviamente deitada) a tela da figura 6.31 com alguns caracteres peculiares do MSX, apesar deles não existirem em sua impressora.

Aliás, você deve ter notado que a própria listagem da figura 6.30 foi tirada com esta técnica.

fig. 6.31 - Escrevendo "em grego" numa impressora não MSX.

Cartuchos

Existem basicamente duas maneiras de executar um programa no MSX. A primeira é carregar o programa através do teclado, cassete ou diskette e a segunda é instalando um cartucho ROM, com um programa previamente gravado em um dos slots disponíveis no micro.

Geralmente, dentro de um cartucho, existe um programa que foi gravado durante sua fabricação e que é executado automaticamente assim que o micro é ligado.

Quando você liga o micro, ele começa a chavear os slots através da porta A da PPI à procura de uma senha que identifica a presença do cartucho. Esta procura é limitada apenas às páginas 1 e 2, pois na página 0 está o BIOS, que não pode ser retirado e na página 3 está a área das variáveis do sistema, que também não pode deixar de estar presente o tempo todo.

Sendo assim, só podemos ter programas em cartucho com, no máximo 32 Kbytes, correspondendo às páginas 1 e 2 e desde que este programa esteja em Assembly.

Com programas em BASIC, que também podem ser gravados em cartucho, existe uma limitação a mais : para ser executado, é necessária a presença do interpretador BASIC residente na página 1. Portanto, o tamanho máximo que um programa em BASIC gravado em cartucho pode ter é de 16 Kbytes e só poderá estar na página 2.

A procura dos cartuchos, realizada assim que o micro é ligado, acontece do slot 0 para o slot 3, sendo rodado o primeiro que é encontrado.

A senha que identifica se existe ou não um cartucho instalado, é necessária para que o micro não confunda a presença do cartucho com outra coisa qualquer. Esta senha é formada pela sequência de bytes nos endereços descritos na figura 6.32.

fig. 6.32 - Sequência de reconhecimento do cartucho.

x000 e x001 - IDENTIFICAÇÃO
x002 e x003 - INICIO
x004 e x005 - EXPANSÃO DE COMANDOS
x006 e x007 - EXPANSÃO DE DISPOSITIVOS
x008 e x009 - TEXTO
x00A a x00F - Reservado

onde "x" é 4 , para os cartuchos na página 1 , ou 8 , para os cartuchos na página 2.

IDENTIFICAÇÃO : Dois bytes que identificam a presença de cartucho cujos valores devem ser &H41 e &H42.

INÍCIO : Endereço da rotina de inicialização das áreas usadas pelo cartucho quando este for utilizado junto com programas em BASIC. Essa rotina "prepara" os periféricos que o cartucho comanda, retornando ao BASIC-MSX com uma instrução RET do Z80. Se o programa do cartucho for um jogo, esse endereço geralmente contém o seu ponto de entrada.

EXPANSÃO DE COMANDOS : Endereço do programa em Assembly a ser executado quando é encontrado o comando CALL em um programa BASIC. Sempre que o BASIC MSX encontra um comando CALL, ele passa o controle para o cartucho e este deve verificar se o argumento do comando CALL pertence ou não a ele, retornando ao BASIC MSX após ter executado os procedimentos correspondentes ou simplesmente sem ter feito nada caso argumento do comando não pertença ao cartucho.

EXPANSÃO DE DISPOSITIVOS : Endereço do programa em Assembly que será executado quando o BASIC MSX encontrar algum periférico que ele não reconheça, permitindo assim que sejam "criados" novos periféricos além dos já existentes de acordo com o cartucho instalado (DRIVE, MODEM, etc.)

TEXTO : Endereço inicial do programa BASIC gravado no cartucho. Este endereço deve apontar para um byte com &H00 e logo em seguida deverá vir o programa em BASIC com os comandos já convertido em tokens da mesma maneira em que se encontram os programas na memória RAM.

Quando o micro é ligado com um ou mais cartuchos, será executado o primeiro que for encontrado com endereço de execução. A procura começa de x002 e vai até x008, com prioridade sobre o número do slot.

APÊNDICES

APÊNDICE I - FILTRO DE IMPRESSORA

A Associação Brasileira de Normas Técnicas (ABNT) padronizou um conjunto de caracteres para impressoras denominado BRASCII que, em sua primeira metade, coincide com o ASCII (para língua inglesa) e com o do MSX (veja figura abaixo). Na sua segunda parte, referente aos caracteres de outras línguas européias (lembre-se, Portugal está na Europa!), a posição já não coincide com a do padrão MSX. Em função disto o Expert 1.1 e o HOTBIT 1.2 têm, em sua ROM, uma sub-rotina denominada " FILTRO BRASCII " que altera o código dos caracteres enviados à impressora. Esta rotina é ativada assim que o micro é ligado. Para desativá-la e ativá-la novamente basta dar um POKE na variável do sistema no endereço &F417 ou alterar o último atributo do comando SCREEN:

Desativa o filtro BRASCII: POKE &HF417,1 ou SCREEN ,,,,1
Ativa o filtro BRASCII: POKE &HF417,0 ou SCREEN ,,,,0

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8																
9																
A		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö		Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö		ø	ù	ú	û	ü	ý	þ	ÿ

APÊNDICE II - SISTEMAS DE NUMERAÇÃO

Sistemas Posicionais

Qual o maior número (quantidade) que se pode representar apenas com os dedos médio e indicador?

Se você for esperto, sua resposta será 3 !

Se você for muito esperto, sua resposta será 8 !

Se, além de muito esperto, você não for um "dedo duro", sua resposta poderá ser 15 !

Isso pode lhe parecer estranho, porém essa estranheza deve-se apenas ao costume. Até o final deste apêndice, você entenderá porquê.

Nós estamos habituados a contar usando o sistema de numeração decimal, onde temos dez algarismos que, dependendo da posição que ocupam no número, tem um valor diferente.

Os dez algarismos são: 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9 .

Vamos verificar como um número qualquer é escrito com esses algarismos (fig. II.1).

fig. II.1 - Importância da posição na notação decimal.

$1\ 9\ 1\ 4\ 1\ 0\ 0\ 9 = 1x10^7 + 9x10^6 + 1x10^5 + 4x10^4 + 1x10^3 + 0x10^2 + 0x10^1 + 9x10^0$

```
! ! ! ! ! ! ! !
! ! ! ! ! ! ! +------------ 9 unidades
! ! ! ! ! ! +-------------- 0 dezenas de unidades
! ! ! ! ! +---------------- 0 centenas de unidades
! ! ! ! +------------------ 1 milhar de unidades
! ! ! +-------------------- 4 dezenas de milhares de unidades
! ! +---------------------- 1 centena de milhares de unidades
! +------------------------ 9 milhões de unidades
+-------------------------- 1 dezena de milhões de unidades
```

Note que o algarismo 1, por exemplo, aparece três vezes no número 19 141 009, entretanto, seu "peso" na quantidade expressa por esse número depende de sua posição. O algarismo 1 mais à esquerda indica 10 milhões de unidades, enquanto o algarismo 1 mais à direita indica apenas 1000 unidades. Algo análogo ocorre com os dois algarismos 0 .

Um sistema de numeração como esse, onde os símbolos têm "pesos" diferentes em função da posição que ocupam num número, é chamado de SISTEMA POSICIONAL.

Um sistema não posicional muito conhecido é o dos algarismos romanos, onde existem apenas 7 símbolos para representar todos os números: I , V , X , L , C , D e M .

Os sistemas não posicionais são claramente desvantajosos em relação aos posicionais quando temos que fazer contas. Para certificar-se disso, tente realizar as somas da figura II.2 .

fig. II.2 - Vantagem dos sistemas posicionais.

```
   Sistema Romano            Sistema Decimal

     MMDCCLIX                  2759
    MDCCCLXVII  +              1867 +
    ----------                 ----

    ----------                 ----
```

Sistema Decimal

Vamos estudar um pouco mais como funciona o sistema decimal. Imagine que você tenha que "contar estrelas". Cada quantidade é representada por um número, conforme a figura II.3 . Observe apenas os números escritos em decimal e em portugues. Não se preocupe com as outras notações, por enquanto.

fig. II.3 - Contando estrelas.

Estrelas	binário	octal	decimal	hexadecimal	em portugues
	0	0	0	0	zero
*	1	1	1	1	um
**	10	2	2	2	dois
***	11	3	3	3	três
****	100	4	4	4	quatro
*****	101	5	5	5	cinco
******	110	6	6	6	seis
*******	111	7	7	7	sete
********	1000	10	8	8	oito
*********	1001	11	9	9	nove
**********	1010	12	10	A	dez
***********	1011	13	11	B	onze
************	1100	14	12	C	doze
*************	1101	15	13	D	treze
**************	1110	16	14	E	quatorze
***************	1111	17	15	F	quinze
****************	10000	20	16	10	desesseis
.	.	.	.	.	.
.	.	.	.	.	.
etc...	etc...	etc...	etc...	etc...	etc...
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.

Quando a quantidade de estrelas é nove, ainda existe um único símbolo para representá-la no sistema decimal. Daí para diante, ao menos dois símbolos são necessários e a posição se torna importante. Ao ver o número 10, sabemos que o 1 significa uma dezena e o 0 significa zero unidades. Com dois algarismos conseguimos representar até o número noventa e nove (99). A partir de então precisamos de três símbolos para representar os números.

Vamos, agora, imaginar que ao invés de 10 algarismos (0 a 9), existissem apenas dois: o 0 e o 1 .

Será que ainda conseguiríamos representar números maiores que 1 ?!

A resposta é SIM! E da mesma forma que escrevemos números maiores que 9 no sistema decimal, isto é, dando pesos diferentes aos algarismos, em função da posição em que eles estiverem!

Sistema Binário

O sistema de numeração que usa apenas dois algarismos (0 e 1) é o Sistema Binário, que também é posicional.

Usando apenas 1 algarismo, o maior número que podemos escrever nesse sistema é 1 . Usando mais de um algarismo, podemos representar números maiores. Veja novamente a figura II.3 e observe como as estrelas

podem ser contadas nesse sistema.

Um número escrito em binário tem uma estrutura muito semelhante aos números escritos em decimal. Por exemplo, o número 29 decimal corresponde ao número 11101 em binário (fig. II.4).

fig. II.4 - O número 29 representado em binário.

```
2 9                         1  1  1  0  1     = 1x2^4 + 1x2^3 + 1x2^2 + 0x2^1 +1x2^0
| |                         |  |  |  |  |
| +- 9x10^0 =  9            |  |  |  |  +--- 1x2^0 =  1
|                           |  |  |  |
+--- 2x10^1 = 20            |  |  |  +------ 0x2^1 =  0
              --            |  |  |
              29            |  |  +--------- 1x2^2 =  4
                            |  |
                            |  +------------ 1x2^3 =  8
                            |
                            +--------------- 1x2^4 = 16
                                                     --
                                                     29
```

No BASIC MSX, um número pode ser escrito em binário desde que se insiram os símbolos &B a sua esquerda: &B10100011

Para obter a forma binária de um número decimal, basta comandar: PRINT BIN$(número)

Agora vamos estudar dois outros sistemas posicionais muito utilizados em micro-computação.

Sistema Octal

Nesse sistema existem apenas 8 algarismos: 0, 1, 2, 3, 4, 5, 6 e 7 . Portanto, o maior número que pode ser representado com apenas um algarismo é sete (7).

O número oito, por exemplo, deve ser escrito assim: 10 .

Como você escreveria o número 33 nesse sistema?

Se a sua resposta for diferente de 41 , você errou!

No BASIC MSX, um número pode ser escrito em octal desde que se insiram os símbolos &O à sua esquerda: &O777 (Cuidado! Não confunda O com 0 !).

Para obter a forma binária de um número decimal, basta comandar: PRINT OCT$(número)

Todos os sistemas posicionais que vimos até agora (decimal, binário e octal) usavam menos que onze símbolos. Veremos, agora, um sistema que usa dezesseis símbolos na representação dos números.

Sistema Hexadecimal

Para escrever números no sistema hexadecimal precisamos de dezesseis símbolos (ou algarismos) diferentes. Os algarismos de 0 a 9 podem ser usados, mas são necessários mais seis símbolos para representar os números 10, 11, 12, 13, 14 e 15 . Geralmente, usa-se as letras A, B, C, D, E e F. Veja mais uma vez a figura II.3 e observe como se pode contar estrelas nesse sistema.

No BASIC MSX, um número pode ser escrito em hexadecimal desde que se insiram os símbolos &H à sua esquerda: &H8FA3

Para obter a forma hexadecimal de um número decimal, basta comandar: PRINT HEX$(número)

Você deve ter notado que quanto maior o número de algarismos disponíveis num dado sistema, mais compacta fica a representação dos nú-

meros. Veja, como exemplo, as representações de um mesmo número nos quatro sistemas posicionais que estudamos (fig.II.5).

fig. II.5 - Representação do número 4095 nos vários sistemas.

Sistema	Representação numérica	número de dígitos
binário	1 1 1 1 1 1 1 1 1 1 1 1	(12 dígitos)
octal	7 7 7 7	(4 dígitos)
decimal	4 0 9 5	(4 dígitos)
hexadecimal	F F F	(3 dígitos)

O número de algarismos disponíveis de um sistema é chamado de BASE. Portanto, quanto maior a base de um sistema, menos dígitos são necessários para representar os números.

Quando, num mesmo texto, representa-se números em vários sistemas, é conveniente especificar a base em que eles estão representados.

Em todo este livro demos preferência, sempre que possível, ao uso do sistema hexadecimal, usando a notação de base própria do BASIC MSX: &H ....

Para terminar, vamos ver como os dedos médio e indicador podem representar 3, 8 ou 15 números diferentes!

O esperto

A resposta do esperto à pergunta inicial deste apêndice é obtida usando os dois dedos como dígitos do sistema binário (levantado=1, abaixado=0). Como a base é 2, com dois dígitos (literalmente!), você consegue 2^2 =4 combinações diferentes, ou seja, conta de 0 a 3 .

O muito esperto

A resposta do muito esperto usa as falanges dos dedos, para dobrá-los (levantado=2 , dobrado=1 , abaixado=0). Como a base é 3, com dois dígitos você consegue 3^2=9 combinações diferentes, contando de 0 até 8 .

O "dedo mole"

Pouquíssimas pessoas conseguem dobrar os dedos no nódulo entre as falanges 1 e 2 sem dobrar também no outro nódulo. Entretanto, isso é possível!

Se você é um dos poucos que conseguem fazer contorcionismos com os dedos, poderá usar um sistema de base 4 para representar números apenas com o médio e o indicador. Dessa forma, com 4^2 =16 possíveis configurações, você consegue contar de 0 a 15 .

Eis aí um bom motivo para você não ser um dedo duro!

Outras respostas

Além das três respostas que citamos, outras mais frequentes podem lhe ocorrer.

Se você é imediatista e confia muito nas pessoas, sua resposta pode ser "DOIS".

Se você desconfia muito das pessoas, sua resposta pode ser "ONZE", e nesse caso, cada dedo representa o número 1 em decimal. Obviamente, essa resposta não permite contar de 0 a 11, mas apenas representar o número onze em decimal! Se você é uma pessoa normal, equilibrada, etc..., sua resposta poderá ser o clássico "Ora! Não me amole!"

APÊNDICE III - INTRODUÇÃO AO ASSEMBLY Z-80

Estrutura do Z-80

Para entender um pouco o que é o ASSEMBLY Z80, é conveniente estudar antes a estrutura interna do microprocessador Z80 (fig. 0.7).

Note que, apesar de poder receber e enviar apenas 8 bits de cada vez (através da barra de dados), o Z80 pode endereçar esses dados através de 16 bits (o que corresponde a 65536 posições diferentes).

O processamento dos dados é realizado em função do conteúdo dos registros de instruções. O Z80 possui um repertório de 158 instruções elementares. Cada uma produz uma saída diferente no circuito de controle de sinais e, em função disso, um processamento específico é realizado.

Para poder trabalhar, o Z80 possui 22 registros internos (de processamento) que servem como área de rascunho, onde são armazenados os valores que serão tratados posteriormente. Os registradores de processamento são essenciais na programação dos microprocessadores, pois quase todas as instruções usam os dados que eles contém, como parâmetros.

De todos os 22 registradores apenas o IX, o IY, o SP e o PC são de 16 bits. Os demais são todos de 8 bits, apesar de, às vezes, serem usados aos pares.

O registrador A é o mais fundamental e, normalmente, ele é usado sozinho, independentemente do registrador F, para acumular o resultado de operações lógicas e aritméticas realizadas pela ALU (Arithmetic Logic Unit). A ALU é uma parte do Z80 que cuida exclusivamente dos cálculos, comparações e operações lógicas.

O registrador F é um indicador de estado, onde cada bit contém alguma informação sobre o resultado da última instrução realizada pela CPU.

Os registradores B,C,D,E,H e L , apesar de serem de oito bits, podem ser usados aos pares (BC , DE e HL), servindo para indicar endereços entre 0 e 65535.

Os registradores I e R são também usados separadamente e para funções um pouco mais complexas. Apenas para não deixarmos o assunto totalmente no ar, basta saber que o registro I cuida dos processos de interrupção do micro, e o registrador R cuida da manutenção de informações na RAM.

O registradores IX e IY, de 16 bits cada, permitem indexar endereços em programas Assembly e o seu funcionamento é semelhante ao do par HL.

O registrador SP (ponteiro de pilha) serve para armazenar um endereço de memória especial, a partir do qual a CPU armazena dados (stack).

Por fim, o registro PC serve para armazenar endereços. Num programa que está sendo executado pelo Z80, cada instrução está armazenada em um ou mais bytes, dependendo de seu tamanho. O registrador PC armazena sempre o endereço do byte que contém a próxima instrução a ser executada.

Instruções do Z80

O Z80 dispõe de 158 instruções elementares. Cada uma delas é apenas uma sequência de bits 0 ou 1.

O conjunto dessas 158 instruções é o vocabulário da Linguagem de Máquina do Z80. Um programa em Linguagem de Máquina é, portanto, uma sequência de bytes onde cada grupo de até quatro bytes representa uma

instrução para o microprocessador.

Quando o Z80 executa uma sequência de instruções, os estágios a serem seguidos são apenas três:

1 - Ler uma instrução;
2 - Executar a instrução lida;
3 - Voltar ao item 1 para ler a próxima instrução.

Algumas instruções fazem com que o Z80 tenha que ler o conteúdo de um certo endereço da memória (ROM ou RAM), ou ainda, que ele tenha que inserir dados em algum endereço da RAM. A referência ao endereço da memória pode ser feita diretamente na instrução ou através dos registros de processamento. Os dois exemplos a seguir ilustram dois tipos de instruções que o Z80 pode realizar:

1) Referência direta a um endereço da memória:
   * leia o conteúdo do endereço 50500 e armazene-o no registro A.
2) Referência indireta a um endereço da memória:
   * leia o conteúdo do endereço contido no par de registros HL e armazene-o no registro A.

É bastante complicada e trabalhosa a programação direta, usando Linguagem de Máquina e, devido a isso, outras linguagens mais simples, como o BASIC, foram criadas. O BASIC, porém, é uma linguagem bastante sofisticada e muito próxima da linguagem humana. Para entender o BASIC, a CPU tem que traduzí-lo para Linguagem de Máquina e isso leva muito tempo. Imagine uma pessoa que só sabe português e tem que ler uma Bíblia escrita em aramaico. Para isso, ela se municia de um bom dicionário aramaico-português e de vários livros de gramática aramaica. Provavelmente, depois de 1 ano, talvez mais, ela tenha terminado a primeira leitura.

Se o texto fosse em portugues, o tempo gasto seria de apenas algumas semanas.

Algo semelhante ocorre quando a CPU tem que traduzir o BASIC para a Linguagem de Máquina. Ela sabe tudo de Linguagem de Máquina, mas não sabe nada de BASIC.

Quanto mais próxima uma linguagem está da linguagem humana, mais alto o seu nível (isso é convencional!). O BASIC é uma linguagem de alto nível. A Linguagem de Máquina é de baixo nível (talvez pela copiosa quantidade de palavrões proferidos pelos que a utilizam!).

As vantagens e desvantagens de cada uma devem ser minuciosamente estudadas ao se decidir usar uma ou outra para fazer um programa. Na figura III.1 você pode comparar o BASIC com a Linguagem de Máquina.

fig. III.1 - BASIC x Linguagem de Máquina.

	BASIC	LINGUAGEM DE MÁQUINA
Programação	fácil	trabalhosa
Execução	lenta	rápida
Memória	muita	pouca
Verificação	rápida	lenta

(Linguagem de Máquina + BASIC) ÷ 2 = ASSEMBLY

Entre a Linguagem de Máquina e o BASIC, existe a linguagem ASSEMBLY (o meio termo feliz!).

Ela mantém a mesma estrutura da Linguagem de Máquina, porém, cada instrução do Z80 é representada por um código mnemônico e não mais por uma sequência de bits. Por exemplo, uma das instruções de máquina do Z80 é a que carrega um dado no registrador A. Em Linguagem de Máquina ela é representada por: 00111110

Em ASSEMBLY, ela é representada por: LD A,n

LD é contração de LoaD, que em ingles significa CARREGAR.

LD A,n significa CARREGAR O REGISTRO A COM O VALOR n .

Obviamente, os programas escritos em ASSEMBLY também precisam ser traduzidos para Linguagem de Máquina antes de serem executados, porém essa tradução é bastante simples e rápida.

Os programas que fazem a tradução do ASSEMBLY para a Linguagem de Máquina são chamados ASSEMBLERS (ou ASSEMBLADORES). Na figura III.2 apresentamos um programa disassembler em BASIC que faz exatamente o contrário: traduz da Linguagem de Máquina para o ASSEMBLY.

Para quem não dispõe de um programa "assemblador", a tradução do Assembly para Linguagem de Máquina pode ser feita com o auxílio de tabelas que correlacionam os mnemônicos com seus códigos (veja, por exemplo, MNEMÔNICOS DO Z-80, desta Editora).

fig. III.2 - Programa Disassembler para o MSX.

```
10000 '--------------------------------------
          DISASSEMBLER MSX
   (C) 1985 - MILTON MALDONADO JR.
              15/12/85
---------------------------------------
10010 FOR I=0 TO 7:READ A$(I),B$(I):NEX
T I:FOR I=0 TO 3:READ C$(I),E$(I):NEXT
I:FOR I=0 TO 7:READ D$(I),F$(I):NEXT I:
READ G$(0),G$(1),G$(2),H$(0),H$(1),H$(2
),I$(0),I$(1),J$(0),J$(1)
10020 WIDTH 39:CLS:INPUT "PRIMEIRO ENDE
RECO ";E
10030 CLS:FOR K=1 TO ?0:C=PEEK(E)
10040 PRINT USING "#####";E;:PRINT "  "
;:GOSUB 10070:IF F)0 THEN I=I+1
10050 PRINT TAB(22);:FOR X=0 TO I-1:I$=
HEX$(PEEK(X+E)):IF LEN(I$)(2 THEN I$="0
"+I$
10060 PRINT I$;" ";:NEXT X:PRINT:E=E+I:
NEXT K:PRINT:PRINT "APERTE QUALQUER TEC
LA (R=REINICIO) ";:I$=INPUT$(1):IF I$="
R" THEN 10020 ELSE 10030
10070 '--------------------------------------
            SEPARA GRUPOS
---------------------------------------
10080 F=0:IF C=221 THEN F=1 ELSE IF C=2
53 THEN F=2
10090 C$(2)="HL":IF F=1 THEN C$(2)="IX"
 ELSE IF F=2 THEN C$(2)="IY"
10100 IF C=203 THEN 10620 ELSE IF C=237
 THEN 10680
10110 IF F)0 AND PEEK(E+1)=203 THEN 106
20
10120 IF F)0 THEN C=PEEK(E+1)
10130 '--------------------------------------
           COMANDOS DIRETOS
---------------------------------------
10140 E$(2)=C$(2)
10150 I=1:IF C=39 THEN PRINT "DAA";ELSE
 IF C=47 THEN PRINT "CPL";ELSE IF C=249
 THEN PRINT "LD SP,";C$(2);ELSE IF C=22
7 THEN PRINT "EX (SP),";C$(2):ELSE IF C
=118 THEN PRINT "HALT";ELSE IF C=201 TH
EN PRINT "RET";ELSE IF C=31 THEN PRINT
"RRA";ELSE 10170
10160 RETURN
10170 IF C=235 THEN PRINT "EX DE,HL";EL
SE IF C=8 THEN PRINT "EX AF,AF'";ELSE I
F C=217 THEN PRINT "EXX";ELSE IF C=233
THEN PRINT "JP (";C$(2);")";ELSE IF C=0
 THEN PRINT "NOP";ELSE IF C=243 THEN PR
INT "DI";ELSE IF C=251 THEN PRINT "EI";
ELSE 10190
10180 RETURN
10190 IF C=7 THEN PRINT "RLCA";ELSE IF
C=15 THEN PRINT "RRCA";ELSE IF C=23 THE
N PRINT "RLA";ELSE IF C=55 THEN PRINT "
SCF";ELSE IF C=63 THEN PRINT "CCF";ELSE
 10210
10200 RETURN
10210 I=2:IF C=211 THEN PRINT "OUT (";:
GOSUB 10490:PRINT "),A";ELSE IF C=219 T
HEN PRINT "IN A,(";:GOSUB 10490:PRINT")
";ELSE 10230
10220 RETURN
10230 IF C=195 THEN PRINT "JP ";ELSE IF
 C=205 THEN PRINT "CALL ";ELSE 10260
10240 PRINT MID$(STR$(PEEK (E+1)+256*PE
EK(E+2)),2,5);:I=3:RETURN
10250 RETURN
10260 IF C=16 THEN PRINT "DJNZ ";ELSE I
F C=24 THEN PRINT "JR ";ELSE 10290
10270 X=PEEK(E+1):IF X(128 THEN Y=E+X+2
 ELSE Y=E+X-254
10280 PRINT MID$(STR$(Y),2,5);:I=2:RETU
RN
10290 IF C)127 AND C(192 THEN X=C\8-16:
Y=C MOD 8:PRINT A$(X);:IF X=0 OR X=1 OR
 X=3 THEN PRINT " A,";ELSE PRINT " ";EL
SE 10310
10300 PRINT B$(Y);:I=1:RETURN
10310 IF C\64)0 THEN 10450
10320 IF C=34 THEN E=E+SGN(F):PRINT "LD
 (";:GOSUB 10240:PRINT "),";C$(2);:ELSE
 IF C=42 THEN E=E+SGN(F):PRINT "LD ";C$
(2);",(";:GOSUB 10240:PRINT ")";:ELSE 1
0340
10330 E=E-SGN(F):RETURN
```

```
10340 IF C=50 THEN PRINT "LD (";:GOSUB
10240:PRINT "),A";ELSE IF C=58 THEN PRI
NT "LD A,(";:GOSUB 10240:PRINT ")";:RET
URN ELSE 10360
10350 I=3:RETURN
10360 X=C MOD 8:IF X=4 THEN PRINT"INC";
ELSE IF X=5 THEN PRINT"DEC";ELSE 10380
10370 Y=C\8:PRINT" ";B$(Y);:I=1:RETURN
10380 X=C MOD 16:IF X=3 THEN PRINT"INC
";ELSE IF X=11 THEN PRINT"DEC ";ELSE IF
 X=9 THEN PRINT"ADD ";C$(2);",";ELSE 10
400
10390 X=C\16:PRINTC$(X);:I=1:RETURN
10400 IF C MOD 8=6 AND F=0 THEN PRINT "
LD ";B$(C\8);",";:GOTO 10490
10410 IF C MOD 16=1 THEN E=E+SGN(F):PRI
NT"LD ";C$(C\16);",";:GOSUB 10240 ELSE
10430
10420 E=E-SGN(F):RETURN
10430 IF C MOD 16=2 THEN PRINT "LD (";C
$(C\16);"),A";ELSE IF C MOD 16=10 THEN
PRINT "LD A,(";C$(C\16);")";ELSE 10450
10440 I=1:RETURN
10450 IF C\32=1 AND C MOD 8=0 THEN PRIN
T "JR ";D$(C\8-4);",";:GOTO 10270
10460 IF C\64<>3 THEN 10560
10470 IF C MOD 8<>6 THEN 10500
10480 I=2:X=C\8-24:PRINTA$(X);" ";:IF X
<2 OR X=3 THEN PRINT"A,";
10490 PRINT MID$(STR$(PEEK(E+1)),2,3);:
RETURN
10500 X=C MOD 8:IF X=1 THEN PRINT "POP
";E$(C\16-12);ELSE IF X=5 THEN PRINT "P
USH ";E$(C\16-12);
10510 IF X=1 OR X=5 THEN I=1:RETURN
10520 IF X=7 THEN PRINT"RST ";HEX$(C-19
9);"H";:I=1:RETURN
10530 IF X=0 THEN PRINT "RET ";D$(C\8-2
4);:I=1:RETURN
10540 IF X=2 THEN PRINT "JP ";ELSE PRIN
T"CALL ";
10550 PRINT D$(C\8-24);",";:GOTO 10240
10560 IF F=0 AND C\64=1 THEN X=(C-64)\8
:PRINT "LD ";B$(X);:X=(C-64) MOD 8:PRIN
T",";B$(X);:I=1:RETURN
10570 I=2:C=PEEK(E+1):IF F>0 AND C MOD
8=6 AND C>63 THEN PRINT "LD ";B$(C\8-8)
;",(";C$(2);"+";:E=E+1:GOSUB 10490:PRIN
T ")"; ELSE 10590
10580 E=E-1:RETURN
10590 IF C\8=14 THEN E=E+1:PRINT "LD ("
;C$(2);"+";:GOSUB 10490:PRINT "),";B$(C
-112); ELSE 10610
10600 GOTO 10580
10610 I=3:IF C=54 THEN PRINT "LD (";C$(
2);"+";:E=E+1:GOSUB 10490:PRINT "),";:E
=E+1:GOSUB 10490:E=E-1:GOTO 10600 ELSE
10840
10620 '-------------------------------------
            COMANDOS APOS CBH
-------------------------------------------
10630 Z=1:IF F>0 THEN Z=3
10640 I=2:C=PEEK(E+Z):IF C<64 AND F=0 T
HEN PRINT F$(C\8);" ";B$(C MOD 8);:RETU
RN
10650 I=3:IF C<64 AND F>0 AND (C-6) MOD
 8=0 THEN PRINT F$(C\8);" (";C$(2);"+";
:E=E+1:GOSUB 10490:PRINT")";:E=E-1:RETU
RN
10660 C=PEEK(E+Z):IF F=0 THEN PRINTG$(C
\64-1);" ";CHR$(48+(C\8) MOD 8);",";B$(
C MOD 8);:I=2:RETURN
10670 IF (C-6) MOD 8=0 THEN PRINTG$(C\6
4-1);" ";CHR$(48+(C\8) MOD 8);",(";C$(2
);"+";:E=E+1:GOSUB 10490:PRINT")";:E=E-
1:RETURN ELSE 10840
10680 '-------------------------------------
            COMANDOS APOS EDH
-------------------------------------------
10690 I=2:C=PEEK(E+1):IF C<64 OR C=221
OR C=253 THEN 10840 ELSE IF C>187 THEN
10800
10700 IF C=70 THEN PRINT "IM 0";ELSE IF
 C=86 THEN PRINT "IM 1";ELSE IF C=94 TH
EN PRINT "IM 2";ELSE IF C=77 THEN PRINT
 "RETI";ELSE IF C=69 THEN PRINT "RETN";
ELSE IF C=103 THEN PRINT "RRD";ELSE IF
C=111 THEN PRINT "RLD";ELSE 10720
10710 RETURN
10720 IF C=71 THEN PRINT "LD I,A";ELSE
IF C=79 THEN PRINT "LD R,A";ELSE IF C=8
7 THEN PRINT "LD A,I";ELSE IF C=95 THEN
 PRINT "LD A,R";ELSE 10740
10730 RETURN
10740 H$(3)="OUT":IF C>175 THEN H$(3)="
OT"
10750 IF C>159 AND C MOD 8<4 THEN PRINT
H$(C MOD 4);I$((C MOD 16)\8);J$((C-160)
\16);:RETURN
10760 IF C\64>1 THEN 10840 ELSE IF C MO
D 8>1 THEN 10800
10770 IF C=112 OR C=113 THEN 10840
10780 IF C MOD 8=0 THEN PRINT "IN ";B$(
C\8-8);",(C)";ELSE PRINT "OUT (C),";B$(
C\8-8);
10790 RETURN
10800 X=C MOD 16:Y=C\16-4:IF X=10 THEN
PRINT"ADC HL,";C$(Y);ELSE IF X=2 THEN P
RINT "SBC HL,";C$(Y);ELSE 10820
10810 RETURN
10820 IF X=3 THEN PRINT "LD (";:E=E+1:G
OSUB 10240:E=E-1:PRINT "),";C$(C\16-4);
ELSE IF X=11 THEN PRINT "LD ";C$(C\16-4
);",(";:E=E+1:GOSUB 10240:E=E-1:PRINT")
";ELSE 10840
10830 I=4:RETURN
10840 PRINT "Z80 ?";:RETURN
10850 '-------------------------------------
            DADOS DA MATRIZ ALFA
-------------------------------------------
10860 DATA ADD,B,ADC,C,SUB,D,SBC,E,AND,
H,XOR,L,OR,(HL),CP,A
10870 DATA BC,BC,DE,DE,HL,HL,SP,AF
10880 DATA NZ,RLC,Z,RRC,NC,RL,C,RR,PO,S
LA,PE,SRA,P,SLI,M,SRL
10890 DATA BIT,RES,SET
10900 DATA LD,CP,IN,I,D, ,R
```

APÊNDICE IV – ROTINAS DO BIOS

Notações utilizadas:

============	XXXX endereço de entrada e nome.
Fun	Função da sub-rotina.
Ent	parâmetros de Entrada.
Ret	parâmetros de Retorno.
Mod	registradores Modificados.
Not	Notas.

Obs: Todos os endereços estão em hexadecimal. Quando é citada uma variável do sistema, seu endereço na RAM está marcado entre parênteses.

```
============ Inicialização ===================
0000  RESET          ;Inicialização do sistema
0004  DW      CGTABL ;Endereço da tabela do
                      gerador de caracteres
0006  DB      VDP.DR ;Endereço do reg de dados
                      do VDP (leitura)
0007  DB      VDP.DW ;Endereço do reg de dados
                      do VDP (escrita)
============ 0008  SYNCHR ====================
Fun   Verifica se o caracter apontado por HL é
      o desejado. Se não for, emite a mensagem
      "Syntax error", caso contrário salta pa-
      ra CHRGTR.
Ent   HL.
Ret   HL - endereço do próximo caractere.
      A  - código do caractere.
      Flag CY = 1 se for número.
      Flag Z  = 1 se for encontrado fim do
                comando.
============ 000C  RDSLT =====================
Fun   Seleciona o slot apropriado de acordo
      com o valor dado pelos registradores e
      lê o conteúdo da memória.
Ent   A - FXXXSSPP
          !   !!!!
          !   !!++- slot primário (0-3).
          !   ++- slot secundário (0-3).
          +- =1 se definido slot secundário.
      HL - endereço da memória a ser lida.
Ret   A  - conteúdo da memória.
Mod   AF, BC, DE.
Not   As interrupções são automaticamente de-
      sabilitadas.
============ 0010  CHRGTR ====================
Fun   Ler o próximo caracter (ou "token") do
      texto do BASIC.
Ent   HL.
Ret   HL - endereço do próximo caractere.
      A  - código do caractere.
      Flag CY = 1 se o dado é número.
      Flag Z  = 1 se for encontrado o fim do
      comando.
============ 0014  WRSLT =====================
Fun   Seleciona o slot apropriado de acordo
      com o valor dado pelos registradores e
      grava um byte na memória.
Ent   A - FXXXSSPP
          !   !!++- slot primário (0-3).
          !   ++- slot secundário (0-3).
          +- = 1 se definido slot secundário.
      HL - Endereço da memória a receber o
           byte.
      E  - byte a ser gravado.
Mod   AF, BC, D
Not   As interrupções são automaticamente
      desabilitadas.
============ 0018  OUTDO =====================
Fun   Envia um dado ao último dispositivo
      referenciado.
Ent   A, PTRFIL (F864), PTRFLG (F416)
============ 001C  CALSLT ====================
Fun   Realiza chamadas entre slots a um ende-
      reço especificado.
Ent   IYH - FXXXSSPP
            !   !!++- slot primário (0-3).
            !   ++- slot secundário (0-3).
            +-= 1 se definido slot secundário.
      IX  - Endereço a ser executado.
Ret   Indeterminado.
Mod   Indeterminado.
Not   As interrup.são automat. desabilit.Nunca
      passar param. pelos regs. A'F', IX e IY.
============ 0020  DCOMPR ====================
Fun   Compara HL com DE.
Ent   HL, DE.
Ret   Flags do Z80.
Mod   AF.
============ 0024  ENASLT ====================
Fun   Seleciona o slot apropriado de acordo
      com o valor dado pelos registradores e o
      mantém permanentemente habilitado.
Ent   A - FXXXSSPP
          !   !!++- slot primário (0-3).
          !   ++- slot secundário (0-3).
          +- = 1 se definido slot secundário.
      HL - endereço da memória a ser habili-
           tada.
Mod   Todos os registradores.
Not   As interrupções são automaticamente de-
      sabilitadas.
============ 0028  GETYPR ====================
Fun   Retorna o tipo de FAC.
Ent   FAC.
Ret   Flags do Z80.
Mod   AF.
============ 002B  ID Bytes. =================
============ 0030  CALLF =====================
Fun   Realiza chamadas de subrotinas entre
      slots.
Ret   Indeterminado.
Mod   Indeterminado.
Not   A sequência de chamada é a seguinte:
      RST 30H
      DB número do slot.
      DW endereço da sub-rotina.
============ 0038  KEYINT ====================
Fun   Realiza os procedimentos necessários às
      interrupções.

As seguintes entradas são usadas para inicia-
lização do BIOS:

============ 003B  INITIO ====================
Fun   Procede a inicialização dos dispositivos
      externos.
Mod   Todos os registradores.
============ 003E  INIFNK ====================
Fun   Inicializa o contúdo das teclas de fun-
      ções.
Mod   Todos os registradores.

As entradas as seguir são usadas para acessar
o VDP (TMS9128):

============ 0041  DISSCR ====================
Fun   Desabilita o gerador de imagens.
Mod   AF, BC.
============ 0044  ENASCR ====================
Fun   Habilita o gerador de imagens.
Mod   AF, BC.
============ 0047  WRTVDP ====================
Fun   Grava um byte num registro do VDP.
Ent   C - Número do registro do VDP.
      B - Byte a ser gravado.
Mod   AF, BC.
============ 004A  RDVRM =====================
Fun   Lê um byte da VRAM.
Ent   HL - Endereço a ser lido.
Ret   A  - Byte lido.
Mod   AF.
```

```
============ 004D  WRTVRM ====================
Fun   Grava um byte na VRAM.
Ent   HL - Endereço da VRAM a receber o byte.
      A  - Byte a ser gravado.
Mod   AF.
============ 0050  SETRD =====================
Fun   Prepara o VDP para leitura.
Ent   HL.
Mod   AF.
============ 0053  SETWRT ====================
Fun   Prepara o VDP para escrita.
Ent   HL.
Mod   AF.
============ 0056  FILVRM ====================
Fun   Preenche uma área da VRAM com um byte
      especificado.
Ent   HL - Endereço de início.
      BC - Tamanho da área a ser preenchida.
      A  - Dado a ser colocado nesta área.
Mod   Todos os registradores.
============ 0059  LDIRMV ====================
Fun   Copia os dados de uma área da VRAM para
      a RAM.
Ent   HL - Endereço da VRAM.
      DE - Endereço da RAM.
      BC - Tamanho da área a ser copiada.
Mod   Todos os registradores.
============ 005C  LDIRVM ====================
Fun   Copia uma área da RAM para a VRAM.
Ent   HL - Endereço da RAM.
      DE - Endereço da VRAM.
      BC - Tamanho da área a ser copiada.
Mod   Todos os registradores.
============ 005F  CHGMOD ====================
Fun   Seleciona o modo de operação do VDP.
Ent   SCRMOD (FCAF).
Mod   Todos os registradores.
============ 0062  CHGCLR ====================
Fun   Muda as cores da tela.
Ent   Cor de frente em FORCLR (F3E9).
      Cor de fundo  em BAKCLR (F3EA).
      Cor de borda  em BDRCLR (F3EB).
Mod   Todos os registradores.
============ 0066  NMI =======================
Fun   Realiza os procedimentos relativos à NMI
      (interrupção não mascarável da CPU).
============ 0069  CLRSPR ====================
Fun   Inicializa todos os sprites com zeros e
      coordenada Y em 209.
Ent   SCRMOD (FCAF).
Mod   Todos os registradores.
============ 006C  INITXT ====================
Fun   Inicializa a tela para SCREEN 0.
Ent   TXTNAM (F3B3), TXTCGP (F3B7)
Mod   Todos os registradores.
============ 006F  INIT32 ====================
Fun   Inicializa a tela para SCREEN 1.
Ent   T32NAM (F3BD), T32CGP (F3C1), T32COL
      (F3BF) T32ATR (F3C3), T32PAT (F3C5)
Mod   Todos os registradores.
============ 0072  INIGRP ====================
Fun   Inicializa a tela para SCREEN 2.
Ent   GRPNAM (F3C7), GRPCGP (F3CB), GRPCOL
      (F3C9) GRPATR (F3CD), GRPPAT (F3CF)
Mod   Todos os registradores.
============ 0075  INIMLT ====================
Fun   Inicializa a tela para SCREEN 3 .
Ent   MLTNAM (F3D1), MLTCGP (F3D5), MLTCOL
      (F3D3) MLTATR (F3D7), MLTPAT (F3D9)
Mod   Todos os registradores.
============ 0078  SETTXT ====================
Fun   Ajusta o VDP para SCREEN 0 .
Ent   TXTNAM (F3B3), TXTCGP (F3B7)
Mod   Todos os registradores.
============ 007B  SETT32 ====================
Fun   Ajusta o VDP para modo texto (24x32).
Ent   T32NAM (F3BD), T32CGP (F3C1), T32COL
      (F3BF) T32ATR (F3C3), T32PAT (F3C5)
Mod   Todos os registradores.
============ 007E  SETGRP ====================
Fun   Ajusta o VDP para SCREEN 2 .
Ent   GRPNAM (F3C7), GRPCGP (F3CB), CRPCOL
      (F3C9) GRPATR (F3CD), GRPPAT(F3CF)
Mod   Todos os registradores.
============ 0081  SETMLT ====================
Fun   Ajusta o VDP para a SCREEN 3.
Ent   MLTNAM (F3D1), MLTCGP (F3D5), MLTCOL
      (F3D3) MLTATR (F3D7), MLTPAT (F3D9)
Mod   Todos os registradores.
============ 0084  CALPAT ====================
Fun   Fornece o endereço da tabela de padrões
      de um sprite.
Ent   A  - Código do sprite.
Ret   HL - Endereço da tabela.
Mod   AF, DE, HL.
============ 0087  CALATR ====================
Fun   Fornece o endereço da tabela de atribu-
      tos de um sprite.
Ent   A  - Código do sprite.
Ret   HL - Endereço da tabela.
Mod   AF, DE, HL.
============ 008A  GSPSIZ ====================
Fun   Fornece o tamanho atual dos sprites.
Ret   A  - Tamanho dos sprites (número de
      bytes).
      Flag CY = 1 se o sprite é 16x16.
Mod   AF
============ 008D  GRPPRT ====================
Fun   Imprime um caractere na tela gráfica.
Ent   A  - Código do caractere.
```

As seguintes entradas são usadas para acessar o PSG:

```
============ 0090  GICINI ====================
Fun   Inicializa o PSG.
Mod   Todos os registradores.
============ 0093  WRTPSG ====================
Fun   Grava um byte num registro do PSG.
Ent   A  - Número do registro.
      E  - Byte a ser gravado.
============ 0096  RDPSG =====================
Fun   Lê um byte de um registro do PSG.
Ent   A  - Número do registro.
Ret   A  - Byte lido.
============ 0099  STRTMS ====================
Fun   Verifica e inicializa comandos prelimi-
      nares para o PLAY.
Mod   Todos os registradores.
```

As entradas seguintes são usadas para entradas e saídas:

```
============ 009C  CHSNS =====================
Fun   Verifica o status do buffer do teclado.
Ret   Flag Z = 1 se há algum caractere no
      buffer.
Mod   AF.
```

```
============ 009F  CHGET =====================
Fun   Aguarda o pressionamento de uma tecla e
      e devolve seu código.
Ret   A  - Código do caractere.
Mod   AF.
============ 00A2  CHPUT =====================
Fun   Envia um caractere ao vídeo.
Ent   A  - Código do caractere.
============ 00A5  LPTOUT ====================
Fun   Envia um caractere para a impressora.
Ent   A  - Código do caractere.
Ret   Flag CY = 1 se operação interrompida.
Mod   F.
============ 00A8  LPTSTT ====================
Fun   Verifica o estado da impressora.
Ret   A  = 255 e flag Z = 0 se a impressora
      está pronta.
      A  =   0 e flag Z = 1 em caso contrário.
Mod   AF
============ 00AB  CNVCHR ====================
Fun   Verifica byte de controle para caractere
      gráfico e faz conversão do código.
Ent   A  - Código do caractere.
Ret   Flag CY = 0 se o byte for de controle
          gráfico.
      Flag CY = 1 e Flag Z = 1 se o código for
          convertido para gráfico.
      Flag CY = 1 e Flag Z = 0 se código não
          convertido.
Mod   AF.
============ 00AE  PINLIN ====================
Fun   Recebe uma linha digitada até o pres-
      sionamento da tecla
      RETURN ou CONTROL+STOP, e armazena a
      linha no buffer.
Ret   HL - Endereço do buffer (menos 1).
      Flag CY = 1 se CONTROL+STOP foi pres-
          sionada.
Mod   Todos os registradores.
============ 00B1  INLIN =====================
Fun   O mesmo que PINLIN exceto quando AUTFLG
      (F6AA) setado.
Ret   O mesmo que em PINLIN.
Mod   Todos os registradores.
============ 00B4  QINLIN ====================
Fun   Imprime " ? " seguido de um espaço e
      salta para INLIN.
Ret   O mesmo que em PINLIN.
Mod   Todos os registradores.
============ 00B7  BREAKX ====================
Fun   Verifica CONTROL+STOP.
Ret   Flag CY = 1 se pressionadas.
Mod   AF.
============ 00BA  ISCNTC ====================
Fun   Verifica SHIFT-STOP.
============ 00BD  CKCNTC ====================
Fun   O mesmo que a rotina ISCNTC, usado pelo
      Basic.
============ 00C0  BEEP ======================
Fun   Gera um bip.
Mod   Todos os registradores.
============ 00C3  CLS =======================
Fun   Limpa a tela.
Mod   AF, BC, DE.
============ 00C6  POSIT =====================
Fun   Posiciona o cursor na posição especi-
      ficada.
Ent   H  - Coluna.
      L  - Linha.
Mod   AF.
============ 00C9  FNKSB =====================
Fun   Mostra o conteúdo das teclas de função
      se necessário.
Ent   FNKFLG (FBCE a FBD7)
Mod   Todos os registradores.
============ 00CC  ERAFNK ====================
Fun   Apaga a apresentação das teclas de fun-
      ção.
Mod   Todos os registradores.
============ 00CF  DSPFNK ====================
Fun   Mostra o conteúdo das teclas de função.
Mod   Todos os registradores.
============ 00D2  TOTEXT ====================
Fun   Força a tela para o modo texto.
Mod   Todos os registradores.
```

As seguintes entradas são usadas para controles de jogos:

```
============ 00D5  GTSTCK ====================
Fun   Retorna o estado do joystick selecio-
      nado.
Ent   A  - Identificador do joystick.
Ret   A  - Direção lida.
Mod   Todos os registradores.
============ 00D8  GTTRIG ====================
Fun   Retorna o estado do botão de disparo se-
      lecionado.
Ent   A - Identificador do botão.
Ret   A = 0 se foi pressionado, senão A = 255.
Mod   AF.
============ 00DB  GTPAD =====================
Fun   Retorna o estado do Touch Pad
      selecionado.
Ent   A  - Identificador Touch Pad.
Ret   A  - Valor lido.
Mod   Todos os registradores.
============ 00DE  GTPDL =====================
Fun   Retorna o valor do Paddle selecionado.
Ent   A  - Identificador do paddle.
Ret   A  - Valor lido.
Mod   Todos os registradores.
```

As seguintes entradas são usadas para acessar o cassete:

```
============ 00E1  TAPION ====================
Fun   Liga o motor e lê o cabeçalho da fita.
Ret   Flag CY = 1 se interrompido.
Mod   Todos os registradores.
============ 00E4  TAPIN =====================
Fun   Lê um byte da fita.
Ret   A  - byte lido.
      Flag CY = 1 se interrompido.
Mod   Todos os registradores.
============ 00E7  TAPIOF ====================
Fun   Encerra a leitura da fita.
============ 00EA  TAPOON ====================
Fun   Liga motor e grava o cabeçalho na fita.
Ent   A  = 0 se o cabeçalho deve ser curto.
      A  = 1 se o cabeçalho deve ser longo.
Ret   Flag CY = 1 se interrompido.
Mod   Todos os registradores.
============ 00ED  TAPOUT ====================
Fun   Grava um byte na fita.
Ent   A  - Byte a ser gravado.
Ret   Flag CY = 1 se interrompido.
Mod   Todos os registradores.
```

```
============ 00F0 TAPOOF ====================
Fun  Encerra a gravação na fita.
============ 00F3 STMOTR ====================
Fun  Aciona o motor do cassete.
Ent  A - 0 desliga o motor.
     A - 1 liga o motor.
     A - 255 inverte o estado do motor.
Mod  AF.
```

As seguintes entradas são usadas para manipular filas:

```
============ 00F6 LEFTQ =====================
Fun  Retorna o número de bytes na fila.
============ 00F9 PUTQ ======================
Fun  Inclui um byte na fila.
```

As seguintes entradas são usadas por GENGRP e ADVGRP:

```
============ 00FC RIGHTC ====================
Fun  Move um pixel para a direita.
============ 00FF LEFTC =====================
Fun  Move um pixel para a esquerda.
============ 0102 UPC =======================
Fun  Move um pixel para cima.
============ 0105 TUPC ======================
Fun  Move um pixel para cima.
============ 0108 DOWNC =====================
Fun  Move um pixel para baixo.
============ 010B TDOWNC ====================
Fun  Move um pixel para baixo.
============ 010E SCALXY ====================
Fun  Coordenada da escala XY.
============ 0111 MAPXYC ====================
Fun  Mapeia a coordenada para o endereço
     físico.
============ 0114 FETCHC ====================
Fun  Busca o endereço físico e padrão da
     máscara.
Ret  Endereço em HL, número da máscara
     em A.
============ 0117 STOREC ====================
Fun  Guarda endereço físico e mascara
     padrão.
Ent  Endereço em HL, número da máscara
     em A.
============ 011A SETATR ====================
Fun  Carrega byte de atributos.
============ 011D READC =====================
Fun  Lê byte de atributo do pixel corrente.
============ 0120 SETC ======================
Fun  Seta o pixel corrente para o atributo
     especificado.
============ 0123 NSETCX ====================
Fun  Seta pixels horizontalmente.
============ 0126 GTASPC ====================
Fun  Retorna razão de aspecto.
Ent  DE, HL.
Ret  DE, HL.

============ 0129 PNTINI ====================
Fun  Inicialização para o PAINT.
============ 012C SCANR =====================
Fun  Procura pixels a direita.
============ 012F SCANL =====================
Fun  Procura pixels a esquerda.
```

A seguir vêm entradas de uso geral:

```
============ 0132 CHGCAP ====================
Fun  Muda o estado da lâmpada do CAPS LOCK.
Ent  A = 0 apaga a lâmpada.
     A ≠ 0 acende a lâmpada.
============ 0135 CHGSND ====================
Fun  Muda o estado do bit de som da porta C.
Ent  A = 0 , bit = 0 .
     A ≠ 0 , bit = 1 .
============ 0138 RSLREG ====================
Fun  Lê byte de controle do slot primário.
Ret  A - Byte lido.
Mod  A
============ 013B WSLREG ====================
Fun  Grava byte de controle do slot primário.
Ent  A - Byte a ser gravado.
============ 013E RDVDP =====================
Fun  Lê o registrador de STATUS do VDP.
Ret  A - Byte lido.
Mod  AF
============ 0141 SNSMAT ====================
Fun  Lê uma coluna do teclado.
Ent  A - Número da linha.
Ret  A - Coluna lida.
Mod  AF
Not  Os bits zerados correspondem às teclas
     pressionadas.
============ 0144 PHYDIO ====================
Fun  Inicialização das áreas de armazenamento
     dos periféricos.
============ 0147 FORMAT ====================
Fun  Inicialização das áreas de armazenamento
     dos periféricos.
============ 014A ISFLIO ====================
Fun  Verifica se está sendo feita operação de
     entrada e saída.
============ 014D OUTDLP ====================
Fun  Envia um byte à impressora.
Ent  A - Byte a ser gravado.
Mod  F
Not  Os caracteres TAB são expandidos.
     Símbolos gráficos são convertidos
     quando a impressora não é padrão MSX.
     Se for interrompida, a mensagem "Device
     I/O error" é apresentada.
============ 0150 GETVCP ====================
Fun  Usado para execução de músicas pelo
     comando PLAY.
============ 0153 GETVC2 ====================
============ 0156 KILBUF ====================
Fun  Limpa o buffer do teclado.
Mod  HL.
============ 0159 CALBAS ====================
Fun  Rotina do CALL do Interpretador BASIC.
Ent  IX - Endereço a ser chamado.
```

Existem a seguir 90 bytes livres (reservados para expansões futuras).

LIVROS DA COLEÇÃO MSX JÁ PUBLICADOS

COLEÇÃO DE PROGRAMAS VOL.I

Uma coletânea de programas, dedicada aos usuários principiantes do MSX. Jogos, músicas, desenhos, apresentados de maneira simples e didática, com instruções de digitação e análise dos programas, praticamente linha por linha. Sugestões de modificação e dicas importantes.

COLEÇÃO DE PROGRAMAS VOL. II

Um conjunto completo de programas, com rotinas em BASIC e em Linguagem de Máquina. Jogos de ação, jogos inteligentes, programas didáticos de química, física, matemática, geografia, biologia, etc. Programas profissionais de estatística e matemática financeira. Programas utilitários para uso da impressora, gravador cassete, etc.

USANDO O DISK DRIVE NO MSX

Neste livro são estudados detalhadamente três sistemas controladores de discos para o MSX: o DSK-BASIC (DISK-BASIC MSX), o HB-DOS (MSX-DOS) e o HB-MCP (CP/M). Além de uma parte inicial, dedicada aos usuários principiantes e textos didáticos sobre cada um dos sistemas em estudo, o livro contém um completo dicionário de comandos e funções com exemplos elucidativos. Indispensável para quem quer conhecer e usar adequadamente os recursos do disk-drive.

PROGRAMAÇÃO AVANÇADA EM MSX

Os recursos poderosíssimos da Linguagem de Máquina do Z80 levados à perfeição no MSX.Como usar os recursos gráficos e sonoros do MSX em ASSEMBLY, como acessar diretamente rotinas do interpretador, como usar as portas de I/O, e muitas outras dicas. O mais profundo e detalhado livro sobre o MSX. Indispensável para os "cobras" da microcomputação.

SITEMA DE DISCO PARA MSX

Um livro sobre o SOLX-DOS e o BASIC DE DISCO, dois sistemas operacionais desenvolvidos pela MICROSOL para seu equipamento CDX-2. Cada comando ou função é comentado detalhadamente e exemplificado com um programa para tornar clara a sua aplicação.

LIVROS EM FASE DE ELABORAÇÃO (no prelo)

Além dos títulos indicados, durante 1987 a COLEÇÃO MSX da EDITORA ALEPH será enriquecida por muitos outros versando sobre:

LINGUAGEM DE MÁQUINA PARA MSX
CURSO DE BASIC MSX
BASIC AVANÇADO (EXPLORANDO O MSX)
COLEÇÃO DE DICAS
ANTOLOGIA DE PROGRAMAS EM LINGUAGEM DE MÁQUINA

Para se manter sempre atualizado sobre nossos lançamentos e receber GRATUITAMENTE nosso boletim informativo com dicas sobre o MSX envie seu nome e endereço completo para:

ALEPH Publicações e Assessoria Pedagógica Ltda
Caixa Postal: 20707 / CEP: 01498 - São Paulo - SP

ESTE LIVRO FOI COMPOSTO NUMA IMPRESSORA MÔNICA PLUS da ELEBRA.

COLEÇÃO MSX

Este livro foi escrito para o único periférico realmente inteligente de um microcomputador: O USUÁRIO!

A implantação do padrão MSX no Brasil gerou uma grande procura de literatura técnica referente aos detalhes do mais completo microcomputador de oito bits atualmente no mercado.

Para suprir as necessidades dos programadores mais exigentes, a EDITORA ALEPH produziu, praticamente a partir do zero, um livro onde os detalhes da arquitetura do MSX são discutidos de maneira completa e didática.

Nele encontramos explicações minuciosas sobre a RAM, a ROM, o BIOS, VARIÁVEIS DO SISTEMA, PSG, VDP, PPI, etc.

Todos esses itens são tratados com uma clareza e com nível de profundidade raramente encontrados mesmo em publicações estrangeiras.

Conseguiu-se, através desta obra, um excelente grau de inteligibilidade, transformando-a no acessório mais necessário para um possuidor de MSX.